Accredited Standards Committee X9 **November 17, 1997**
Title: X9-Financial Services
Accredited by the
American National Standards Institute

Working Draft

AMERICAN NATIONAL STANDARD
X9.62-199x, Public Key Cryptography For The Financial Services Industry:
The Elliptic Curve Digital Signature Algorithm (ECDSA)©

Notice -- This document is a draft document. It has not yet been processed through the consensus procedures of X9 and ANSI.
Many changes which may greatly affect its contents can occur before this document is completed. The working group may not be held responsible for the contents of this document.
Implementation or design based on this draft is at the risk of the user. No advertisement or citation implying compliance with a "Standard" should appear as it is erroneous and misleading to so state.
Copies of the draft proposed American National Standard will be available from the X9 Secretariat when the document is finally announced for two months public comment. Notice of this announcement will be in the trade press.

Secretariat: American Bankers Association
Standards Department
1120 Connecticut Ave., N.W.
Washington, DC 20036

Foreword
(Informative)

Business practice has changed with the introduction of computer-based technologies. The substitution of electronic transactions for their paper-based predecessors has reduced costs and improved efficiency. Trillions of dollars in funds and securities are transferred daily by telephone, wire services, and other electronic communication mechanisms. The high value or sheer volume of such transactions within an open environment exposes the financial community and its customers to potentially severe risks from accidental or deliberate alteration, substitution or destruction of data. This risk is compounded by interconnected networks, and the increased number and sophistication of malicious adversaries.

Some of the conventional "due care" controls used with paper-based transactions are unavailable in electronic transactions. Examples of such controls are safety paper which protects integrity, and handwritten signatures or embossed seals which indicate the intent of the originator to be legally bound. In an electronic-based environment, controls must be in place that provide the same degree of assurance and certainty as in a paper environment. The financial community is responding to these needs.

This Standard, X9.62-199x, *Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm* (*ECDSA*), defines a technique for generating and validating digital signatures.

This standard describes a method for digital signatures using the elliptic curve analog of the Digital Signature Algorithm (DSA) (ANSI X9.30-1).

Elliptic curve systems are public-key (asymmetric) cryptographic algorithms that are typically used:

- to create digital signatures (in conjunction with a hash algorithm), and
- to establish secret keys securely for use in symmetric-key cryptosystems.

The primary advantage of elliptic curve systems is their apparent high cryptographic strength relative to key size. The attractiveness of elliptic curve cryptosystems may increase relative to other public-key cryptosystems as computing power improvements warrant a general increase in key size. The shorter key sizes may result in significantly shorter certificates and system parameters. These potential advantages manifest themselves in many ways, including storage efficiencies, bandwidth savings, and computational efficiencies. The computational efficiencies may lead in turn to higher speeds, power efficiency, code size reductions, or a combination thereof. These potential efficiencies are particularly beneficial in applications such as:

- high volume transaction systems,
- wireless communications,
- hand-held computing (e.g., personal digital assistants),
- broadcast communications, and
- smart cards,

where bandwidth, processing capacity, power availability or storage are constrained.

When implemented with proper controls, the techniques of this Standard provide:

- data integrity, and
- non-repudiation of the message origin and the message contents.

Additionally, when used in conjunction with a Message Identifier[1], the techniques of this Standard provide the capability of detecting duplicate transactions. It is the Committee's belief that the proper implementation of this Standard should also contribute to the enforceability of some legal obligations.

The use of this Standard, together with appropriate controls, may have a legal effect, including the apportionment of liability for erroneous or fraudulent transactions and the satisfaction of statutory or contractual "due care" requirements. The legal implications associated with the use of this Standard may be affected by case law and legislation, including the Uniform Commercial Code Article 4A on Funds Transfers (Article 4A).

The details of Article 4A address, in part, the use of commercially reasonable security procedures and the effect of using such procedures on the apportionment of liability between a customer and a bank. A security procedure is provided by Article 4A-201 "for the purpose of (i) verifying that a payment order or communication amending or canceling a payment order originated is that of the customer, or (ii) detecting an error in the transmission or the content of the payment order or communication." The commercial reasonableness of a security procedure is determined by the criteria established in Article 4A-201.

While the techniques specified in this Standard are designed to maintain the integrity of financial messages and provide the service of non-repudiation, the Standard does not guarantee that a particular implementation is secure. It

[1] ANSI X9.9-1986, Financial Institution Message Authentication (Wholesale).

is the responsibility of the financial institution to put an overall process in place with the necessary controls to ensure that the process is securely implemented. Furthermore, the controls should include the application of appropriate audit tests in order to verify compliance with this Standard.

Suggestions for the improvement or revision of this Standard are welcome. They should be sent to the X9 Committee Secretariat, American Bankers Association, 1120 Connecticut Avenue, N.W., Washington D.C. 20036.

This Standard was processed and approved for submittal to ANSI by the Accredited Standards Committee on Financial Services, X9. Committee approval of the Standard does not necessarily imply that all the committee members voted for its approval. At the time that this Standard was approved, the X9 Committee had the following members:

Harold Deal, Chairman
Alice Droogan, Vice Chairman
Cynthia Fuller, Secretariat

Organization Represented	**Representative**
[to be furnished]	

The X9F subcommittee on Data and Information Security had the following members:

Glenda Barnes, Chairman

Organization Represented	**Representative**
[to be furnished]	

The X9F1 working group which developed this standard had the following members:

M. Blake Greenlee, Chairman

Organization Represented	**Representative**
[to be furnished]	

Contents

Figures

Tables

X9.62-199x
Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)

1. Scope

This Standard defines methods for digital signature (signature) generation and verification for the protection of messages and data using the Elliptic Curve Digital Signature Algorithm (ECDSA).
The ECDSA shall be used in conjunction with the hash function defined in ANSI X9.30-1993, *Part* 2, *Secure Hash Algorithm* (*SHA*-1)(*revised*). In addition, this ECDSA Standard provides the criteria for the generation of public and private keys that are required by the algorithm and the procedural controls required for the secure use of the algorithm.

2. Definitions, Abbreviations, Symbols and Notation

2.1. Definitions and Abbreviations

addition rule	An *addition rule* describes the addition of two elliptic curve points P_1 and P_2 to produce a third elliptic curve point P_3. (See Sections C.3 and C.4.)
asymmetric cryptographic algorithm	A cryptographic algorithm that uses two related keys, a public key and a private key; the two keys have the property that, given the public key, it is computationally infeasible to derive the private key.
basis	A representation of the elements of the finite field F_{2^m}. Two special kinds of basis are *polynomial basis* and *normal basis.* (See Section C.2.)
binary polynomial	A polynomial whose coefficients are in the field F_2.
bit string	A bit string is an ordered sequence of 0's and 1's.
certificate	The public key and identity of an entity together with some other information, rendered unforgeable by signing the certificate with the private key of the certifying authority which issued that certificate. In this Standard the term certificate shall mean a public-key certificate.
Certification Authority (CA)	A Center trusted by one or more entities to create and assign certificates.
characteristic 2 finite field	A finite field containing 2^m elements, where $m \geq 1$ is an integer.
compressed form	Octet string representation for a point using the point compression technique described in Section 4.4.1. (See also Section 4.4.2.)
cryptographic hash	A (mathematical) function which maps values from a large (possibly very large) domain into a smaller range. The function satisfies the following properties: 1. it is computationally infeasible to find any input which maps to any pre-specified output; 2. it is computationally infeasible to find any two distinct inputs which map to the same output.
cryptographic key (key)	A parameter that determines the operation of a cryptographic function such as:

1. the transformation from plaintext to ciphertext and vice versa,
2. the synchronized generation of keying material,
3. a digital signature computation or validation.

cryptography — The discipline which embodies principles, means and methods for the transformation of data in order to hide its information content, prevent its undetected modification, prevent its unauthorized use, or a combination thereof.

cryptoperiod — The time span during which a specific key is authorized for use or in which the keys for a given system may remain in effect.

digital signature — The result of a cryptographic transformation of data which, when properly implemented, provides the services of:

1. origin authentication,
2. data integrity, and
3. signer non-repudiation.

EC — Elliptic curve.

ECDSA — Elliptic curve analog of the Digital Signature Algorithm (DSA) (ANSI X9.30, Part 1).

elliptic curve — An *elliptic curve* is a set of points specified by 2 parameters a and b, which are elements of a field F_q. The elliptic curve is said to be defined over F_q; F_q is sometimes called the *underlying field*.

If $q = p$ is an odd prime, $p > 3$, (so the field is F_p), then the Weierstrass equation defining the curve is of the form $y^2 = x^3 + ax + b$, where $((4a^3 + 27b^2) \bmod p) \neq 0$. If q is a power of 2 (so the field is F_{2^m}), then the Weierstrass equation defining the curve is of the form $y^2 + xy = x^3 + ax^2 + b$, where $b \neq 0$.

elliptic curve key pair — Given particular elliptic curve parameters, an *elliptic curve key pair* consists of an elliptic curve private key and the corresponding elliptic curve public key.

elliptic curve private key — Given particular elliptic curve parameters, an *elliptic curve private key, d,* is a statistically unique and unpredictable integer in the interval $[1, n - 1]$, where n is the prime order of the base point P.

elliptic curve public key — Given particular elliptic curve parameters, and an elliptic curve private key d, the corresponding *elliptic curve public key, Q,* is the elliptic curve point $Q = dP$, where P is the base point. Note that Q will never equal $\mathcal{O}$, since $1 \leq d \leq n - 1$.

elliptic curve parameters — These parameters specify an underlying field F_q, the equation of an elliptic curve over F_q, an elliptic curve point P of prime order, the order n of P, and the cofactor h. (See Section 5.1.)

elliptic curve point — If E is an elliptic curve defined over a field F_q, then an *elliptic curve point P* is either:

1. a pair of field elements (x_P, y_P) (where $x_P, y_P \in F_q$) such that the values $x = x_P$ and $y = y_P$ satisfy the equation defining E, or

2. a special point $\mathcal{O}$ called the *point at infinity*.

hash value	The result of applying a cryptographic hash function to a message.
hybrid form	Octet string representation for both the compressed and uncompressed forms of an elliptic curve point. (See Section 4.4.2.)
irreducible polynomial	A binary polynomial $f(x)$ is *irreducible* if it does not factor as a product of two binary polynomials, each of degree less than the degree of $f(x)$.
key	See cryptographic key.
keying material	The data (e.g., keys, certificates and initialization vectors) necessary to establish and maintain cryptographic keying relationships.
message	The data to be signed.
message identifier (MID)	A field which may be used to identify a message. Typically, this field is a sequence number.
non-repudiation	This service provides proof of the integrity and origin of data which can be verified by a third party.
normal basis (NB)	A type of basis that can be used to represent the elements of the finite field F_{2^m}. (See Section C.2.3.)
octet	An *octet* is a bit string of length 8. An octet is represented by a hexadecimal string of length 2. The first hexadecimal digit represents the four leftmost bits of the octet, and the second hexadecimal digit represents the four rightmost bits of the octet. For example, 9*D* represents the bit string 10011101. An octet also represents an integer in the interval [0, 255]. For example, 9*D* represents the integer 157.
octet string	An octet string is an ordered sequence of octets.
optimal normal basis (ONB)	A type of normal basis that can be used to represent the elements of the finite field F_{2^m}. (See Section C.2.4.)
order of a curve	The *order of an elliptic curve E* defined over the field F_q is the number of points on *E*, including $\mathcal{O}$. This is denoted by $\#E(F_q)$.
order of a point	The *order of a point P* is the smallest positive integer *n* such that $nP = \mathcal{O}$ (the point at infinity).
owner	The entity whose identity is associated with a private/public key pair.
pentanomial	A polynomial of the form $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1,$ where $1 \le k_1 < k_2 < k_3 \le m-1.$
pentanomial basis (PPB)	A type of polynomial basis that can be used to represent the elements of the finite field F_{2^m}. (See Section C.2.2.)
point compression	Let *P* be a point (x_P, y_P) on an elliptic curve *E* defined over a field F_q. *Point compression* allows the point *P* to be represented using x_P and a single additional bit $\tilde{y}_P$ derived from x_P and y_P. If *q* is a prime number, then $\tilde{y}_P$ is equal to the rightmost bit of y_P. If *q* is a power of 2, then $\tilde{y}_P$

	is 0 if $x_P = 0$; if $x_P \neq 0$, then $\tilde{y}_P$ is equal to the rightmost bit of the field element $y_P \bullet x_P^{-1}$. The point compression technique is described in Section 4.4.1.
point compression octet (*PC*)	The rightmost bit of *PC* shall be equal to the value of $\tilde{y}_P$ if the compressed or hybrid forms are used; otherwise it shall be 0. The second and third rightmost bits shall (respectively) be 1 and 0 if the compressed form is used. The second and third rightmost bits shall (respectively) be 0 and 1 if the uncompressed form is used. The second and third rightmost bits shall (respectively) be 1 and 1 if the hybrid form is used. The three rightmost bits shall be 0 to indicate the point at infinity. The remaining five bits of the octet shall be set to 0.
polynomial basis (PB)	A type of basis that can be used to represent the elements of the finite field F_{2^m}. (See Section C.2.1.)
prime finite field	A finite field containing *p* elements, where *p* is an odd prime number.
private key	In an asymmetric (public) key system, that key of an entity's key pair which is known only by that entity.
public key	In an asymmetric key system, that key of an entity's key pair which is publicly known.
reduction polynomial	The irreducible binary polynomial $f(x)$ of degree *m* that is used to determine a polynomial basis representation of F_{2^m}.
scalar multiplication	If *k* is a positive integer, then *kP* denotes the point obtained by adding together *k* copies of the point *P*. The process of computing *kP* from *P* and *k* is called *scalar multiplication*.
The Secure Hash Algorithm, Revision 1 (SHA-1)	SHA-1 implements a hash function which maps messages of a length less than 2^{64} bits to hash values of a length which is exactly 160 bits.
signatory	The entity that generates a digital signature on data.
statistically unique	For the generation of *n*-bit quantities, the probability of two values repeating is less than or equal to the probability of two *n*-bit random quantities repeating.
trinomial	A polynomial of the form $x^m + x^k + 1$, where $1 \leq k \leq m-1$.
trinomial basis (TPB)	A type of polynomial basis that can be used to represent the elements of the finite field F_{2^m}. (See Section C.2.2.)
uncompressed form	Octet string representation for an uncompressed elliptic curve point. (See Section 4.4.2.)
verifier	The entity that verifies the authenticity of a digital signature.
x-coordinate	The *x-coordinate* of an elliptic curve point, $P = (x_P, y_P)$, is x_P.

y-coordinate	The *y-coordinate* of an elliptic curve point, $P = (x_P, y_P)$, is y_P.

2.2. Symbols and Notation

$[a, b]$	The interval of integers between and including a and b.
$\lceil a \rceil$	Ceiling: the smallest integer $\geq a$. For example, $\lceil 5 \rceil = 5$ and $\lceil 5.3 \rceil = 6$.
$\lfloor a \rfloor$	Floor: the largest integer $\leq a$. For example, $\lfloor 5 \rfloor = 5$ and $\lfloor 5.3 \rfloor = 5$.
a mod n	The unique remainder r, $0 \leq r \leq n - 1$, when integer a is divided by n. For example, 23 mod 7 = 2.
B	MOV threshold. A positive integer B such that taking discrete logarithms over F_{q^B} is at least as difficult as taking elliptic curve logarithms over F_q. For this Standard, B shall be ≥ 20.
d	EC private key.
E	An elliptic curve.
$E(F_q)$	The set of all points on an elliptic curve E defined over F_q and including the point at infinity $\mathcal{O}$.
$\#E(F_q)$	If E is defined over F_q, then $\#E(F_q)$ denotes the number of points on the curve (including the point at infinity $\mathcal{O}$). $\#E(F_q)$ is called the order of the curve E.
F_{2^m}	The finite field containing 2^m elements, where m is a positive integer.
F_p	The finite field containing p elements, where p is a prime.
F_q	The finite field containing q elements. For this Standard, q shall either be an odd prime number (p) or a power of 2 (2^m).
h	$h = \#E(F_q)/n$, where n is the order of the base point P. h is called the *cofactor*.
k	Per-message secret value. For this Standard, k shall be a statistically unique and unpredictable integer in the interval $[1, n - 1]$.
l	The length of a field element in octets; $l = \lceil t / 8 \rceil$.
l_{max}	Upper bound on the largest prime divisor of the cofactor h.
$\log_2 x$	The logarithm of x to the base 2.
m	The *degree* of the finite field F_{2^m}.
M	Message to be signed.
M’	Message as received.
MID	Message Identifier.

mod	modulo.
mod n	arithmetic modulo n.
n	The order of the point P. For this Standard, n shall be a prime number. n is the primary security parameter. In general, as n increases, the security of ECDSA also increases. See Appendix D for more information.
$\mathcal{O}$	A special point on an elliptic curve, called the point at infinity. This is the additive identity of the elliptic curve group.
p	An odd prime number.
P	A distinguished point (x_P, y_P) on an elliptic curve called the *base point*.
Q	EC public key.
r_{min}	Lower bound on the desired (prime) order n of the base point P. For this Standard r_{min} shall be $>2^{160}$.
t	The length of a field element in bits; $t = \lceil \log_2 q \rceil$. In particular, if $q = 2^m$, then a field element in F_{2^m} can be represented as a bit string of bit length $t = m$.
T	In the probabilistic primality test (Section F.2.1), the number of independent test rounds to execute. For this Standard T shall be ≥50.
Tr	Trace function. (See Section G.1.5.)
$\|\|X\|\|$	Length in octets of the octet string X.
$X\|\|Y$	Concatenation of two strings X and Y. X and Y are either both bit strings, or both octet strings.
$X \oplus Y$	Bitwise exclusive-or of two bit strings X and Y of the same bit length.
$\tilde{y}_P$	The representation of the y-coordinate of a point P when point compression is used.
Z_p	The set of integers modulo p, where p is an odd prime number.

3. Application

3.1. General

When information is transmitted from one party to another, the recipient may desire to know that the information has not been altered in transit. Furthermore, the recipient may wish to be certain of the originator's identity. Both of these services can be provided through the appropriate use of the ECDSA.

A digital signature is an electronic analog to a written signature and may be used in proving to a third party that the information was, in fact, signed by the claimed originator. Unlike their written counterparts, digital signatures also verify the integrity of information. Digital signatures may also be generated for stored data and programs so that the integrity of the data and programs may be verified at any later time.

3.2. The Use of the ECDSA Algorithm

The ECDSA is used by a *signatory* to generate a digital signature on data and by a *verifier* to verify the authenticity of the signature. Each signatory has a public and private key. The private key is used in the signature generation process, and the public key is used in the signature verification process. For both signature generation and verification, the message, *M*, is compressed by means of the Secure Hash Algorithm (SHA) specified in ANSI X9.30-199x, *Part* 2, *Secure Hash Algorithm* (*SHA*-1) (*Revised*) prior to the signature generation and verification process. An adversary, who does not know the private key of the signatory, cannot generate the correct signature of the signatory. In other words, signatures cannot be forged. However, by using the signatory's public key, anyone can verify a validly signed message.

The user of the public key of a private/public key pair requires assurance that the public key represents the owner of that key pair. That is, there must be a binding of an owner's identity and the owner's public key. This binding may be certified by a mutually trusted party. This may be accomplished by using a Certification Authority which generates a certificate in accordance with ANSI X9.57, *Certificate Management*.

This Standard provides the capability of detecting duplicate messages and preventing the replay of messages when the signed message includes:

1. the identity of the intended recipient, and
2. a MID.

The MID shall not repeat during the cryptoperiod of the underlying private/public key pair. Appendix A of ANSI X9.9-1986 provides information on the use of unique MIDs.

3.3. Control of Keying Material

In the ECDSA asymmetric cryptographic system, the integrity of signed data is dependent upon:

1. the prevention of unauthorized disclosure, use, modification, substitution, insertion, and deletion of the private key, *d*, and the per-message value, *k*, and
2. the prevention of unauthorized modification, substitution, insertion, and deletion of elliptic curve parameters for the ECDSA computation procedures.

Therefore, if *d* is disclosed, the integrity of any message signed using that *d* can no longer be assured. Similarly, the values for the elliptic curve parameters must be protected.

3.4. Appendices

The Appendices to this Standard provide additional requirements and information on the ECDSA and its implementation.

Appendix	Contents	Normative (N) or Informative (I)
A	An Overview of Elliptic Curve Systems	I
B	The Elliptic Curve Analog of the DSA (ECDSA)	I
C	Mathematical Background	I
D	Security Considerations	I
E	Tables of Trinomials, Pentanomials and Optimal Normal Bases	I
F	Normative Number-Theoretic Algorithms	N
G	Informative Number-Theoretic Algorithms	I
H	Examples of ECDSA and Sample Curves	I
I	Small Examples of the ECDSA	I

4. Mathematical Conventions

4.1. Finite Field Arithmetic

This section describes the representations that shall be used for the elements of the underlying finite field F_q. Implementations with different internal representations that produce equivalent results are allowed. Mathematics background and examples are provided in Appendix C.

4.1.1. The Finite Field F_p

The elements of the finite field F_p are the integers $\{0, 1, 2, ..., p - 1\}$.

i) The multiplicative identity element is the integer 1.

ii) The zero element is the integer 0.

iii) Addition of field elements is performed modulo p.

iv) Multiplication of field elements is performed modulo p.

4.1.2. The Finite Field F_{2^m}

The elements of the finite field F_{2^m} are the bit strings of bit length m.

There are numerous methods for representing the elements of the finite field F_{2^m}. Two such methods are a *polynomial basis* (PB) *representation* (see Section C.2.1) and a *normal basis* (NB) *representation* (see Section C.2.3). A *trinomial basis* (TPB) and a *pentanomial basis* (PPB) are special types of polynomial bases; these bases are described in Section 4.1.3. An *optimal normal basis* (ONB) is a special type of normal basis; these bases are described in Section 4.1.4.

One of TPB, PPB, or ONB shall be used as the basis for representing the elements of the finite field F_{2^m} in implementing this Standard, as described in Sections 4.1.3 and 4.1.4 . The choice of representation does not affect the security of the ECDSA. Section G.2.3 describes one method for converting the elements of F_{2^m} from one representation to another.

4.1.3. Trinomial and Pentanomial Basis Representation

A *polynomial basis representation* of F_{2^m} over F_2 is determined by an irreducible polynomial $f(x)$ of degree m over F_2; $f(x)$ is called the *reduction polynomial*. The set of polynomials $\{x^{m-1}, x^{m-2}, \ldots, x, 1\}$ forms a basis of F_{2^m} over F_2, called a *polynomial basis*. The elements of F_{2^m} are the bit strings of a bit length which is exactly m. A typical element $a \in F_{2^m}$ is represented by the bit string $a = (a_{m-1}a_{m-2} \ldots a_1a_0)$, which corresponds to the polynomial $a(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_1x + a_0$.

i) The multiplicative identity element (1) is represented by the bit string (00…001).

ii) The zero element (0) is represented by the bit string of all 0's.

iii) Addition of two field elements is accomplished by XORing the bit strings.

iv) Multiplication of field elements a and b is defined as follows. Let $r(x)$ be the remainder polynomial obtained upon dividing the product of the polynomials $a(x)$ and $b(x)$ by $f(x)$. Then $a \bullet b$ is defined to the bit string corresponding to the polynomial $r(x)$.

See Section C.2.1 for further details and an example of a polynomial basis representation.

A *trinomial* over F_2 is a polynomial of the form $x^m + x^k + 1$ where $1 \le k \le m-1$. A *pentanomial* over F_2 is a polynomial of the form $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, where $1 \le k_1 < k_2 < k_3 \le m-1$.

A *trinomial basis representation* of F_{2^m} is a polynomial basis representation determined by an irreducible trinomial $f(x) = x^m + x^k + 1$ of degree m over F_2. Such trinomials only exist for certain values of m. Table E-2 in Appendix E lists an irreducible trinomial of degree m over F_2 for each m, $160 \le m \le 2000$, for which an irreducible trinomial of degree m exists. For each such m, the table lists the smallest k for which $x^m + x^k + 1$ is irreducible over F_2.

A *pentanomial basis representation* of F_{2^m} is a polynomial basis representation determined by an irreducible pentanomial $f(x) = x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$ of degree m over F_2. Such pentanomials exist for all values of $m \ge 4$. Table E-3 in Appendix E lists an irreducible pentanomial of degree m over F_2 for each m, $160 \le m \le 2000$, for which an irreducible trinomial of degree m does not exist. For each such m, the table lists the triple (k_1, k_2, k_3) for which (i) $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$ is irreducible over F_2; (ii) k_1 is as small as possible; (iii) for this particular value of k_1, k_2 is as small as possible; and (iv) for these particular values of k_1 and k_2, k_3 is as small as possible.

a. Rules for selecting a trinomial and pentanomial basis representation

1. If a polynomial basis representation is used for F_{2^m} where there exists an irreducible trinomial of degree m over F_2, then the reduction polynomial shall be an irreducible trinomial of degree m over F_2. To maximize the chances for interoperability, the reduction polynomial used should be $x^m + x^k + 1$ for the smallest possible k. Examples of such polynomials are given in Table E-2 in Appendix E.

2. If a polynomial basis representation is used for F_{2^m} where there does not exist an irreducible trinomial of degree m over F_2, then the reduction polynomial shall be an irreducible pentanomial of degree m over F_2. To maximize the chances for interoperability, the reduction polynomial used should be $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, where (i) k_1 is as small as possible; (ii) for this particular value of k_1, k_2 is as small as possible; and (iii) for these particular values of k_1 and k_2, k_3 is as small as possible. Examples of such polynomials are given in Table E-3 in Appendix E.

4.1.4. Optimal Normal Basis Representation

Optimal normal bases (ONB) over F_2 only exist in F_{2^m} for certain values of m. Table E-1 in Appendix E lists all the values of m, $160 \le m \le 2000$, for which the field F_{2^m} has an ONB over F_2. There are two kinds of ONB, called type I ONB and type II ONB. The difference between the two types of optimal normal bases is in the mathematical formulas which define them; these mathematical formulas are presented in Section 4.1.4.a.1.[2] In the case that a finite field F_{2^m} has both a type I and a type II ONB, the type II ONB shall be used.

The elements of the finite field F_{2^m} are the bit strings of bit length which is exactly m. A typical element $a \in F_{2^m}$ is represented by the bit string $a = (a_0 a_1 \ldots a_{m-2} a_{m-1})$.

i) The multiplicative identity element (1) is represented by the bit string of all 1's.

ii) The zero element (0) is represented by the bit string of all 0's.

iii) Addition of two field elements is accomplished by XORing the bit strings.

iv) Multiplication of field elements is facilitated by the pre-computation of the *product terms* λ_{ij}, $0 \le i, j \le m-1$; a constructive definition of the product terms is given below. An example of computing

[2] Reference [31] provides mathematical definitions of type I and type II ONB.

the product terms is given in Section C.2.4. Note that there are other ways to determine these product terms.

4.1.4.a. Setup for multiplication in F_{2^m} using an optimal normal basis representation

A polynomial $g(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_1x + a_0$ (where each a_i is in the binary field F_2) will be represented by the bit string:

$a_{m-1}a_{m-2}...a_1a_0$

of length m. For example, the polynomial $g(x) = x^4 + x^3 + 1$ is represented by the bit string 11001 of length $m = 5$.

1. If F_{2^m} only has a type I ONB, then let $f(x) = x^m + x^{m-1} + ... + x^2 + x + 1$. Otherwise, if F_{2^m} has a type II ONB, then compute $f(x) = f_m(x)$ using the following recursive formulas:

   $f_0(x) = 1$,
   $f_1(x) = x + 1$,
   $f_{i+1}(x) = xf_i(x) + f_{i-1}(x)$, $i \geq 1$.

   At each stage, the coefficients of the polynomials $f_i(x)$ are reduced modulo 2. Hence $f(x)$ is a polynomial of degree m with coefficients in F_2. The set of polynomials $\{x, x^2 \bmod f(x), x^{2^2} \bmod f(x), \ldots, x^{2^{m-1}} \bmod f(x)\}$ forms a basis of F_{2^m} over F_2, called a *normal basis*.
2. Construct the $m \times m$ matrix A whose i^{th} row, $0 \leq i \leq m - 1$, is the bit string corresponding to the polynomial $x^{2^i} \bmod f(x)$. (The rows and columns of A are indexed by the integers from 0 to $m - 1$.) The entries of A are elements of F_2.
3. Determine the inverse matrix A^{-1} of A.
4. Construct the $m \times m$ matrix T whose i^{th} row, $0 \leq i \leq m - 1$, is obtained as follows.
   a. Compute the polynomial $x \bullet x^{2^i} \bmod f(x)$ and let the corresponding bit string be denoted v;
   b. the i^{th} row of T is the bit string vA^{-1}.
5. Determine the *product terms* λ_{ij}, for $0 \leq i, j \leq m - 1$, as follows:

   $\lambda_{ij} = T(j-i, -i)$.

   Here, $T(i, j)$ denotes the (i, j)-entry of T with indices reduced modulo m. Each *product term* λ_{ij} is an element of F_2.

   It is the case that $\lambda_{0j} = 1$ for precisely one j, $0 \leq j \leq m - 1$, and that for each i, $1 \leq i \leq m - 1$, $\lambda_{ij} = 1$ for precisely two distinct j, $0 \leq j \leq m - 1$. Hence, exactly $2m - 1$ of the m^2 entries of the matrix T are 1, the rest being 0. (As $2m - 1$ nonzero entries is the minimum possible, this normal basis is called an *optimal normal basis.*)

4.1.4.b. Multiplication in F_{2^m} using an optimal normal basis

Let $a = (a_0a_1a_2\ldots a_{m-1})$ and $b = (b_0b_1b_2\ldots b_{m-1})$ be two elements in F_{2^m}. Then the product of a and b is the field element $c = (c_0c_1c_2\ldots c_{m-1})$, where the coefficients c_k are computed using the formula:

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_{i+k}\, b_{j+k}\, \lambda_{ij},\ 0 \leq k \leq m-1,$$

where all subscripts are reduced modulo m. The quantities λ_{ij} are the product terms defined in Section 4.1.4.a.

An example of field arithmetic using an optimal normal basis representation is given in Section C.2.4.

4.1.5. Fields F_{2^m} Having Both ONB and TPB.

Table E-4 in Appendix E lists the values of m, $160 \leq m \leq 2000$, for which the field F_{2^m} has both an optimal normal basis representation (ONB) and a trinomial basis representation (TPB).

4.2. Data Representation

The data types in this Standard are octet strings, integers, field elements and elliptic curve points. Figure 1 provides a cross-reference for the sections defining conversions between data types that shall be used in the algorithms specified in this Standard. The number on a line is the section number where the conversion technique is specified. Examples of conversions are provided in Section H.1.

4.2.1. Integer-to-Octet-String Conversion

Input: A non-negative integer x, and the intended length k of the octet string satisfying: $2^{8k} > x$.

Output: An octet string M of length k octets.

1. Let $M_1, M_2, ..., M_k$ be the octets of M from leftmost to rightmost.
2. The octets of M shall satisfy:

$$x = \sum_{i=1}^{k} 2^{8(k-i)} M_i.$$

4.2.2. Octet-String-to-Integer Conversion

Input: An octet string M of length k octets.

Output: An integer x.

1. Let $M_1, M_2, ..., M_k$ be the octets of M from leftmost to rightmost.
2. M shall be converted to an integer x satisfying:

$$x = \sum_{i=1}^{k} 2^{8(k-i)} M_i.$$

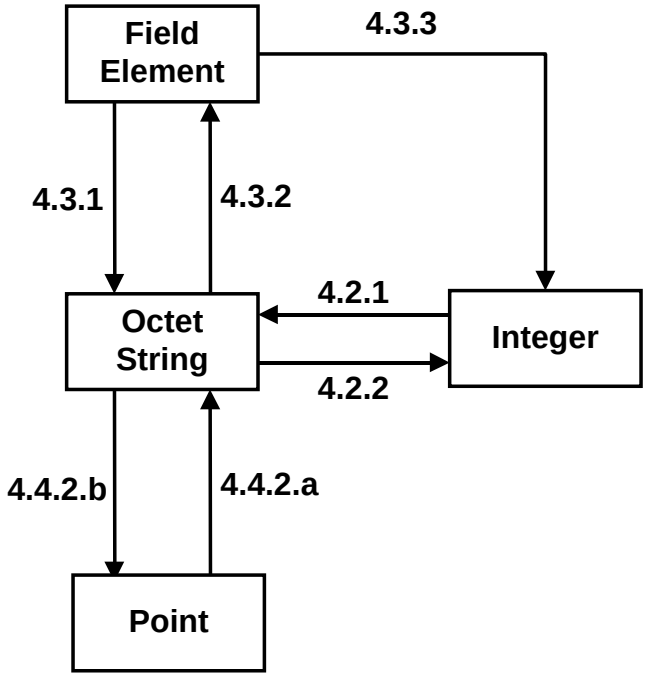

Figure 1 Data Types and Conversion Conventions.

4.3. Finite Field Element Representations

4.3.1. Field-Element-to-Octet-String Conversion

Input: An element α in the field F_q.

Output: An octet string S of length $l = \lceil t / 8 \rceil$ octets, where $t = \lceil \log_2 q \rceil$.

1. If q is an odd prime, then α must be an integer in the interval $[0, q - 1]$; α shall be converted to an octet string of length l octets using the technique specified in Section 4.2.1.

2. If $q = 2^m$, then α must be a bit string of length m bits. Let $s_1, s_2, \ldots, s_m$ be the bits of α from leftmost to rightmost. Let $S_1, S_2, \ldots, S_l$ be the octets of S from leftmost to rightmost. The rightmost bit s_m shall become the rightmost bit of the last octet S_l, and so on through the leftmost bit s_1, which shall become the $(8l - m + 1)^{th}$ bit of the first octet S_1. The leftmost $(8l - m)$ bits of the first octet S_1 shall be zero.

4.3.2. Octet-String-to-Field-Element Conversion

Input: An indication of the field F_q used, and an octet string S of length $l = \lceil t / 8 \rceil$ octets, where $t = \lceil \log_2 q \rceil$.

Output: An element α in F_q.

1. If q is an odd prime, then convert S to an integer α using the technique specified in Section 4.2.2. It is an error if α does not lie in the interval $[0, q - 1]$.

2. If $q = 2^m$, then α shall be a bit string of length m bits. Let $s_1, s_2, \ldots, s_m$ be the bits of α from leftmost to rightmost. Let $S_1, S_2, \ldots, S_l$ be the octets of S from leftmost to rightmost. The rightmost bit of the last octet S_l shall become the rightmost bit s_m, and so on through the $(8l - m + 1)^{th}$ bit of the first octet S_1, which shall become the leftmost bit s_1. The leftmost $(8l - m)$ bits of the first octet S_1 are not used.

4.3.3. Field-Element-to-Integer Conversion

Input: An element α in the field F_q.

Output: An integer x.

1. If q is an odd prime then $x = \alpha$ (no conversion is required).

2. If $q = 2^m$, then α must be a bit string of length m bits. Let $s_1, s_2, \ldots, s_m$ be the bits of α from leftmost to rightmost. α shall be converted to an integer x satisfying:

$$x = \sum_{i=1}^{m} 2^{(m-i)} s_i.$$

4.4. Elliptic Curve Point Representations

An elliptic curve point P (which is not the point at infinity $\mathcal{O}$) is represented by two field elements, the x-coordinate of P and the y-coordinate of P: $P = (x_P, y_P)$. The point can be represented compactly by storing only the x-coordinate x_P and a certain bit $\tilde{y}_P$ derived from the x-coordinate x_P and the y-coordinate y_P. The next subsection describes the technique that shall be used to recover the full y-coordinate y_P from x_P and $\tilde{y}_P$, if this point compression technique is used.

4.4.1. Point Compression Technique

4.4.1.a. Point compression technique (elliptic curves over F_p)

Let $P = (x_P, y_P)$ be a point on the elliptic curve $E : y^2 = x^3 + ax + b$ defined over a prime field F_p. Then $\tilde{y}_P$ is defined to be the rightmost bit of y_P.

When the x-coordinate x_P of P and the bit $\tilde{y}_P$ are provided, then y_P can be recovered as follows.

1. Compute the field element $\alpha = x_P^3 + ax_P + b \bmod p$.

2. Compute a square root β of $\alpha \bmod p$. (See Section G.1.4.) It is an error if the output of G.1.4 is "no square roots exist".

3. If the rightmost bit of β is equal to $\tilde{y}_P$, then set $y_P = \beta$. Otherwise, set $y_P = p - \beta$.

4.4.1.b. Point compression technique (elliptic curves over F_{2^m})

Let $P = (x_P, y_P)$ be a point on the elliptic curve $E : y^2 + xy = x^3 + ax^2 + b$ defined over a field F_{2^m}. Then $\tilde{y}_P$ is defined to be 0 if $x_P = 0$; if $x_P \neq 0$, then $\tilde{y}_P$ is defined to be the rightmost bit of the field element $y_P \bullet x_P^{-1}$.

When the x-coordinate x_P of P and the bit $\tilde{y}_P$ are provided, then y_P can be recovered as follows.

1. If $x_P = 0$, then $y_P = b^{2^{m-1}}$. (y_P is the square root of b in F_{2^m}.)
2. If $x_P \neq 0$, then do the following:

   1.1 Compute the field element $\beta = x_P + a + bx_P^{-2}$ in F_{2^m}.

   1.2 Find a field element z such that $z^2 + z = \beta$ using the algorithm described in Section G.1.6. It is an error if the output of G.1.6 is "no solutions exist".

   1.3 Let $\tilde{z}$ be the rightmost bit of z.

   1.4 If $\tilde{y}_P \neq \tilde{z}$, then set $z = z + 1$.

   1.5 Compute $y_P = x_P \bullet z$.

4.4.2. Conversions

The octet string representation of the point at infinity $\mathcal{O}$ shall be a single zero octet $PC = 00$.

An elliptic curve point (x_1, y_1) which is not the point at infinity shall be represented as an octet string in one of the following three forms:

1. compressed form.
2. uncompressed form.
3. hybrid form (an uncompressed point which also includes $\tilde{y}_1$).

4.4.2.a. Point-to-Octet-String conversion

Input: An elliptic curve point (x_1, y_1), not the point at infinity.

Output: An octet string PO of length $l + 1$ octets if the compressed form is used, or of length $2l + 1$ octets if the uncompressed or hybrid form is used. ($l = \lceil (\log_2 q) / 8 \rceil$.)

1. Convert the field element x_1 to an octet string X_1. (See Section 4.3.1.)
2. If the compressed form is used, then do the following:

   2(a) Compute the bit $\tilde{y}_1$. (See Section 4.4.1.)

   2(b) Assign the value 02 to the single octet PC if $\tilde{y}_1$ is 0, or the value 03 if $\tilde{y}_1$ is 1.

   2(c) The result is the octet string $PO = PC \| X_1$.
3. If the uncompressed form is used, then do the following:

   3(a) Convert the field element y_1 to an octet string Y_1. (See Section 4.3.1.)

   3(b) Assign the value 04 to the single octet PC.

   3(c) The result is the octet string $PO = PC \| X_1 \| Y_1$.
4. If the hybrid form is used, then do the following:

   4(a) Convert the field element y_1 to an octet string Y_1. (See Section 4.3.1.)

   4(b) Compute the bit $\tilde{y}_1$. (See Section 4.4.1.)

   4(c) Assign the value 06 to the single octet if $\tilde{y}_1$ is 0, or the value 07 if $\tilde{y}_1$ is 1.

   4(d) The result is the octet string $PO = PC \| X_1 \| Y_1$.

4.4.2.b. Octet-String-to-Point conversion

Input: An octet string PO of length $l + 1$ octets if the compressed form is used, or of length $2l + 1$ octets if the uncompressed or hybrid form is used ($l = \lceil (\log_2 q) / 8 \rceil$), and field elements a, b which define an elliptic curve over F_q.

Output: An elliptic curve point (x_1, y_1), not the point at infinity.

1. If the compressed form is used, then parse PO as follows: $PO = PC \| X_1$, where PC is a single octet, and X_1 is an octet string of length l octets. If

uncompressed or hybrid form is used, then parse *PO* as follows: $PO = PC \| X_1 \| Y_1$, where *PC* is a single octet, and X_1 and Y_1 are octet strings each of length *l* octets.

2. Convert X_1 to a field element x_1. (See Section 4.3.2.)
3. If the compressed form is used, then do the following:

   3(a) Verify that *PC* is either 02 or 03. (It is an error if this is not the case.)

   3(b) Set the bit $\tilde{y}_1$ to be equal to 0 if *PC* = 02, or 1 if *PC* = 03.

   3(c) Convert (x_1, $\tilde{y}_1$) to an elliptic curve point (x_1, y_1). (See Section 4.4.1.)
4. If the uncompressed form is used, then do the following:

   4(a) Verify that *PC* is 04. (It is an error if this is not the case.)

   4(b) Convert Y_1 to a field element y_1. (See Section 4.3.2.)
5. If the hybrid form is used, then do the following:

   5(a) Verify that *PC* is either 06 or 07. (It is an error if this is not the case.)

   5(b) Convert Y_1 to a field element y_1. (See Section 4.3.2.)
6. If *q* is a prime, verify that $y_1^2 \equiv x_1^3 + ax_1 + b \pmod{p}$. (It is an error if this is not the case.)

   If $q = 2^m$, verify that $y_1^2 + x_1 y_1 = x_1^3 + ax_1^2 + b$ in F_{2^m}. (It is an error if this is not the case.)
7. The result is (x_1, y_1).

5. The Elliptic Curve Digital Signature Algorithm (ECDSA)

This section specifies the process of elliptic curve parameter generation and their validation, key generation and validation, and the calculations needed for generation and verification of signatures. Equivalent computations that result in identical output are allowed.

5.1. Elliptic Curve Parameter Generation and Validation

Elliptic curve parameters may either be common to several key pairs (*common elliptic curve parameters*) or specific to one key pair (*specific elliptic curve parameters*). The elliptic curve parameters may be public; the security of the system does not rely on these parameters being secret. The elliptic curve may either be randomly generated in a verifiable manner or in any manner the user chooses (see Section F.3.2). Two cases are distinguished:

1. when the underlying field is F_p (*p* an odd prime), and
2. when the underlying field is F_{2^m}.

Note that *n* is the primary security parameter. In general, as *n* increases, the security of ECDSA also increases. See Appendix D for more information.

5.1.1. Elliptic Curve Parameters and their Validation Over F_p

5.1.1.a. Elliptic curve parameters over F_p

Elliptic curve parameters over F_p shall consist of the following parameters:

a. A field size $q = p$ which defines the underlying finite field F_q, where $p > 3$ shall be a prime number;

b. (Optional) A bit string SEED of length at least 160 bits, if the elliptic curve was randomly generated in accordance with Section F.3.3;

c. Two field elements *a* and *b* in F_q which define the equation of the elliptic curve *E*: $y^2 = x^3 + ax + b$;

d. Two field elements x_P and y_P in F_q which define a point $P = (x_P, y_P)$ of prime order on *E* (note that $P \neq \mathcal{O}$);

e. The order *n* of the point *P* (it must be the case that $n > 2^{160}$); and

f. The cofactor $h = \#E(F_q)/n$.

Section F.3.2 specifies the method that shall be used for generating an elliptic curve E over F_p and the point P of order n.

5.1.1.b. Elliptic curve parameter validation over F_p

The following conditions shall be verified by the generator of the system parameters. These conditions may alternately be verified by a user of the system parameters.

a. Verify that $q = p$ is an odd prime number. (Use one of the primality tests in Sections F.2.1 and F.2.3.)

b. Verify that a, b, x_P and y_P are integers in the interval $[0, p - 1]$.

c. If the elliptic curve was randomly generated in accordance with Section F.3.3, verify that SEED is a bit string of length at least 160 bits, and that a and b were suitably derived from SEED. (See Section F.3.4.b.)

d. Verify that $(4a^3 + 27b^2) \not\equiv 0 \pmod{p}$.

e. Verify that $y^2_P \equiv x^3_P + ax_P + b \pmod{p}$.

f. Verify that n is prime and that $n > 2^{160}$. (See Sections F.2.1 and F.2.3.)

g. Verify that $nP = \mathcal{O}$. (See Section G.3.2.)

h. If $n > 4\sqrt{p}$ then compute $h' = \lfloor (\sqrt{p} + 1)^2 / n \rfloor$, and verify that $h = h'$.

i. Verify that the MOV and Anomalous conditions hold. (See Section F.1.)

If any of the above verifications fail, then reject the elliptic curve parameters.

Notes:

1. The cofactor h is not used in ECDSA, but is included here for compatibility with ANSI X9.63 where it is needed.

2. Step h of Section 5.1.1.b (and also step h of Section 5.1.2.b) verifies that the value of the cofactor h is correct in the case that $n > 4\sqrt{q}$. In the case that $n \le 4\sqrt{q}$, there are efficient methods for verifying the cofactor h, but these methods are not described here for the following reason: elliptic curves used in practice usually have $n \approx q$, and hence the condition $n > 4\sqrt{q}$ will be satisfied.

5.1.2. Elliptic Curve Parameters and their Validation Over F_{2^m}

5.1.2.a. Elliptic curve parameters over F_{2^m}

Elliptic curve parameters over F_{2^m} shall consist of the following parameters:

a. A field size $q = 2^m$ which defines the underlying finite field F_q, an indication of the basis used to represent the elements of the field (TPB, PPB or ONB), and a reduction polynomial of degree m over F_2 if the basis used is a TPB or PPB;

b. (Optional) A bit string SEED of length at least 160 bits, if the elliptic curve was randomly generated in accordance with Section F.3.3;

c. Two field elements a and b in F_q which define the equation of the elliptic curve E: $y^2 + xy = x^3 + ax^2 + b$;

d. Two field elements x_P and y_P in F_q which define a point $P = (x_P, y_P)$ of prime order on E (note that $P \neq \mathcal{O}$);

e. The order n of the point P (it must be the case that $n > 2^{160}$); and

f. The cofactor $h = \#E(F_q)/n$.

Section F.3.2 specifies the method that shall be used for generating an elliptic curve E over F_{2^m} and the point P of order n.

5.1.2.b. Elliptic curve parameter validation over F_{2^m}

The following conditions shall be verified by the generator of the system parameters. These conditions may alternately be verified by a user of the system parameters.

a. Verify that $q = 2^m$ for some m. If the basis used is a TPB, verify that the reduction polynomial is a trinomial and is irreducible over F_2 (see Table E-2 or Section G.2.4). If the basis used is a PPB, verify that an irreducible trinomial of degree m does not exist, and that the reduction polynomial is a pentanomial and is irreducible over F_2 (see Table E-3 or Section G.2.4). If the basis used is an ONB, verify that an optimal normal basis exists for F_{2^m} (see Table E-1 or Section G.1.8).

b. Verify that a, b, x_P and y_P are bit strings of length m bits.

c. If the elliptic curve was randomly generated in accordance with F.3.3, verify that SEED is a bit string of length at least 160 bits, and that b was suitably derived from SEED. (See Section F.3.4.a.)

d. Verify that $b \neq 0$.

e. Verify that $y^2_P + x_P y_P = x^3_P + a x^2_P + b$ in F_{2^m}.

f. Verify that n is prime and that $n > 2^{160}$. (See Sections F.2.1 and F.2.3.)

g. Verify that $nP = \mathcal{O}$. (See Section G.3.2.)

h. If $n > 4\sqrt{q}$ then compute $h' = \lfloor (\sqrt{q} + 1)^2 / n \rfloor$, and verify that $h = h'$.

i. Verify that the MOV and Anomalous conditions hold. (See Section F.1.)

If any of the above verifications fail, then reject the elliptic curve parameters.

5.2. Key Generation and Validation

5.2.1. Key Generation

Given particular elliptic curve parameters, an elliptic curve key pair shall be generated by performing the following operations:

1. Select a statistically unique and unpredictable integer d in the interval [1,n-1]. It is acceptable to use a random or pseudorandom number. If a pseudorandom number is used, it shall be generated using one of the procedures of Secton F.4. If a pseudorandom number is used, optional information to store with the private key are the seed values and the particular pseudorandom generation method used. Storing this optional information helps allow auditing of the key generation process.

   If a pseudorandom generation method is used, the seed values used in the generation of d may be determined by internal means, be supplied by the caller, or both—this is an implementation choice. In all cases, the seed values have the same security requirements as the private key value. That is, they must be protected from unauthorized disclosure and be unpredictable.

2. Compute the point $Q = (x_Q, y_Q) = dP$. (See Section G.3.2.)

3. The key pair is (Q, d), where Q is the public key, and d is the private key.

5.2.2. Key Validation

Given particular elliptic curve parameters and a public key Q, the public key may be verified as follows.

1. Verify that Q is not the point at infinity $\mathcal{O}$.

2. Verify that x_Q and y_Q are elements in the field F_q, where x_Q and y_Q are the x and y coordinates of Q, respectively. (That is, verify that x_Q and y_Q are integers in the interval [0, p - 1] in the case that $q = p$ is an odd prime, or that x_Q and y_Q are bit strings of length m bits in the case that $q = 2^m$.)

3. If $q = p$ is an odd prime, verify that $y_Q^2 \equiv x_Q^3 + a x_Q + b \pmod{p}$. If $q = 2^m$, verify that $y_Q^2 + x_Q y_Q = x_Q^3 + a x_Q^2 + b$ in F_{2^m}.

4. Verify that $nQ = \mathcal{O}$. (See Section G.3.2.)

If any one of the above verifications fail, then reject the public key.

Note: If there is more than one public key available, it may also be checked that no two public keys are the same.

5.3. Signature Generation

This section describes the ECDSA signature generation process.

The signature generation process consists of message digesting, elliptic curve computations, and modular computations.

The input to the signature process is:

1. the message, M, of an arbitrary length, which is represented by a bit string.
2. elliptic curve parameters: q, a, b, $P = (x_P, y_P)$, and n.
3. an elliptic curve private key, d.

5.3.1. Message Digesting

Compute the hash value $e = H(M)$ using the hash function SHA-1 as specified in ANSI X9.30-1993, Part 2. e is represented as an integer with a length of 160 bits.

5.3.2. Elliptic Curve Computations

1. Select a statistically unique and unpredictable integer k in the interval $[1,n-1]$. It is acceptable to use a random or pseudorandom number. If a pseudorandom number is used, it shall be generated using one of the procedures of Section F.4.

   (a) If a pseudorandom generation method is used, the seed values used in the generation of k may either be determined by internal means, be supplied by the caller, or both—this is an implementation choice. In all cases, the seed values have the same security requirements as the private key value. That is, they must be protected from unauthorized disclosure and be unpredictable.

   (b) If the implementation allows a seed supplied by the caller, then the physical security of the device is of utmost importance. This is because if an adversary gained access to the signature generation device and were able to generate a signature with a seed of its choice for the per-message secret k, then the adversary could easily recover the private key.

2. Compute the elliptic curve point $(x_1, y_1) = kP$. (See Section G.3.2.)
3. Convert the field element x_1 to an integer $\overline{x}_1$, as described in Section 4.3.3.
4. Set $r = \overline{x}_1 \bmod n$.
5. If $r = 0$, then go to step 1.

5.3.3. Modular Computations

1. Compute $s = k^{-1}(e + dr) \bmod n$. (See Section G.1.2. for one method to compute $k^{-1} \bmod n$.)
2. If $s = 0$, then go to step 1 of Section 5.3.2.

5.3.4. The Signature

The signature for M shall be the two integers, r and s, as computed in Sections 5.3.2 and 5.3.3.

Note: As an optional security check (to guard against malicious or non-malicious errors in the signature generation process), the signer may verify that (r, s) is indeed a valid signature for message M using the signature verification process described in Section 5.4.

5.4. Signature Verification

This section describes the ECDSA signature verification process.

The signature verification process consists of message digesting, elliptic curve computations, and signature checking.

The input to the signature verification process is:

1. the received message, *M'*, represented as a bit string.
2. the received signature for *M*, represented as the two integers, *r'* and *s'*.
3. elliptic curve parameters: $q, a, b, P = (x_P, y_P)$, and n.
4. an elliptic curve public key, Q.

5.4.1. Message Digesting

Compute the hash value $e = H(M')$ using the hash function SHA-1 as specified in ANSI X9.30-1993, Part 2. e is represented as an integer with a length of 160 bits.

5.4.2. Elliptic Curve Computations

1. If r' is not an integer in the interval [1, n -1], then reject the signature.
2. If s' is not an integer in the interval [1, n -1], then reject the signature.
3. Compute $c = (s')^{-1} \bmod n$. (See Section G.1.2.)
4. Compute $u_1 = ec \bmod n$ and $u_2 = r'c \bmod n$.
5. Compute the elliptic curve point $(x_1, y_1) = u_1P + u_2Q$ (see Section G.3.2). (If $u_1P + u_2Q$ is the point at infinity, then reject the signature.)

5.4.3. Signature Checking

1. Convert the field element x_1 to an integer $\bar{x}_1$, as described in Section 4.3.3.
2. Compute $v = \bar{x}_1 \bmod n$.
3. If $r' = v$, then the signature is verified, and the verifier has a high level of confidence that the received message was sent by the party holding the secret key d corresponding to Q.

   If r' does not equal v, then the message may have been modified, the message may have been incorrectly signed by the signatory, or the message may have been signed by an impostor. The message shall be considered invalid.

6. ASN.1 Syntax

This section provides the syntax for elliptic curve parameters and keys according to Abstract Syntax Notation One (ASN.1). While it is not required that elliptic curve parameters and keys be represented with ASN.1 syntax, if they are so represented, then their syntax shall be as defined here. These ASN.1 definitions shall be encoded using Distinguished Encoding Rules (DER).

The object identifier ansi-X9.62 represents the root of the tree containing all object identifiers defined in this Standard, and has the following value:

```
ansi-X9-62 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) 10045 }
```

6.1. Syntax for Finite Field Identification

This section provides the abstract syntax definitions for the finite fields defined in this Standard.

A finite field shall be identified by a value of type FieldID:

```
FieldID { FIELD-ID:IOSet } ::= SEQUENCE {
   fieldType   FIELD-ID.&id({IOSet}),
   parameters  FIELD-ID.&Type({IOSet}{@fieldType})  OPTIONAL
}

FieldTypes FIELD-ID ::= {
```

```
   { Prime-p            IDENTIFIED BY prime-field } |
   { Characteristic-two IDENTIFIED BY characteristic-two-field },
   ...
}

FIELD-ID ::= TYPE-IDENTIFIER
```

Note:

FieldID is a parameterized type composed of two components, fieldType and parameters. These components are specified by the fields &id and &Type, which form a template for defining sets of information objects, instances of the class FIELD-ID. This class is based on the useful information object class TYPE-IDENTIFIER, described in X.681 Annex A. In an instance of FieldID, "fieldType" will contain an object identifier value that uniquely identifies the type contained in "parameters". The effect of referencing "fieldType" in both components of the fieldID sequence is to tightly bind the object identifier and its type.

The information object set FieldTypes is used as the single parameter in a reference to type FieldID. FieldTypes contains two objects followed by the extension marker ("..."). Each object, which represents a finite field, contains a unique object identifier and its associated type. The values of these objects define all of the valid values that may appear in an instance of fieldID. The extension marker allows backward compatibility with future versions of this standard which may define objects to represent additional kinds of finite fields.

The object identifier id-fieldType represents the root of a tree containing the object identifiers of each field type. It has the following value:

```
id-fieldType OBJECT IDENTIFIER ::= { ansi-X9-62 fieldType(1) }
```

The object identifiers prime-field and characteristic-two-field name the two kinds of fields defined in this Standard. They have the following values:

```
prime-field OBJECT IDENTIFIER ::= { id-fieldType 1 }
characteristic-two-field OBJECT IDENTIFIER ::= { id-fieldType 2 }
Prime-p ::= INTEGER    -- Field size p
Characteristic-two ::= SEQUENCE {
   m                   INTEGER,                       -- Field size 2^m
   basis               CHARACTERISTIC-TWO.&id({BasisTypes}),
   parameters          CHARACTERISTIC-TWO.&Type({BasisTypes}{@basis})
}


BasisTypes CHARACTERISTIC-TWO::= {
   { NULL         IDENTIFIED BY onBasis } |
   { Trinomial    IDENTIFIED BY tpBasis } |
   { Pentanomial  IDENTIFIED BY ppBasis },
   ...
}


Trinomial ::= INTEGER

Pentanomial ::= SEQUENCE {
   k1  INTEGER,
```

```
   k2  INTEGER,
   k3  INTEGER
}
```

CHARACTERISTIC-TWO ::= TYPE-IDENTIFIER

The object identifier id-characteristic-two-basis represents the root of a tree containing the object identifiers for each type of basis for the characteristic-two finite fields. It has the following value:

```
id-characteristic-two-basis OBJECT IDENTIFIER ::= {
                           characteristic-two-field basisType(3) }
```

The object identifiers onBasis, tpBasis and ppBasis name the three kinds of basis for characteristic-two finite fields defined in this Standard. They have the following values:

```
onBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 1 }
tpBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 2 }
ppBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 3 }
```

Notes:

1. For the finite field F_p, where p is an odd prime, the parameter p is specified by a value of type Prime-p.
2. For the finite field F_{2^m}, the components of Characteristic-two are:
   - m: degree of the field.
   - basis: the type of representation used (ONB, TPB, or PPB).

   3. For a trinomial basis representation of F_{2^m}, Trinomial specifies the integer k where $x^m + x^k + 1$ is the reduction polynomial.

   4. For a pentanomial basis representation of F_{2^m}, the components k1, k2, and k3 of Pentanomial specify the integers k_1, k_2, and k_3, respectively, where $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$ is the reduction polynomial.

6.2. Syntax for Finite Field Elements and Elliptic Curve Points

A finite field element shall be represented by a value of type FieldElement:

```
FieldElement ::= OCTET STRING
```

The value of FieldElement shall be the octet string representation of a field element following the conversion routine in Section 4.3.1.

An elliptic curve point shall be represented by a value of type ECPoint:

```
ECPoint  ::= OCTET STRING
```

The value of ECPoint shall be the octet string representation of an elliptic curve point following the conversion routine in Section 4.4.2.a.

6.3. Syntax for Elliptic Curve Parameters

This section provides syntax for representing elliptic curve parameters.

Elliptic curve parameters shall be represented by a value of type ECParameters:

```
ECParameters  ::= SEQUENCE {
      version INTEGER { ecpVer1(1) } (ecpVer1),
      fieldID FieldID { {FieldTypes} },
      curve Curve,
      base ECPoint,
      order INTEGER,
      cofactor INTEGER,
      ... }
```

```
Curve  ::= SEQUENCE {
      a  FieldElement,
      b  FieldElement,
      seed  BIT STRING OPTIONAL }
```

The components of type `ECParameters` have the following meanings:

- `version` specifies the version number of the elliptic curve parameters. It shall have the value 1 for this version of the Standard. The notation above creates an `INTEGER` named `ecpVer1` and gives it a value of one. It is used to constrain `version` to a single value.
- `fieldID` identifies the finite field over which the elliptic curve is defined. Finite fields are represented by values of the parameterized type `FieldID`, constrained to the values of the objects defined in the information object set `FieldTypes`.
- `curve` specifies the coefficients *a* and *b* of the elliptic curve *E*. Each coefficient shall be represented as a value of type `FieldElement`, an `OCTET STRING. seed` is an optional parameter used to derive the coefficients of a randomly generated elliptic curve.
- `base` specifies the base point *P* on the elliptic curve. The base point shall be represented as a value of type `ECPoint`, an `OCTET STRING.`
- `order` specifies the order *n* of the base point.
- `cofactor` is the integer $h = \#E(F_q)/n$.

6.4. Syntax for Public Keys

This section provides the syntax for the public keys defined in this Standard.
A public key may be represented in a variety of ways using `ASN.1` syntax. When a public key is represented as the `X.509` type `SubjectPublicKeyInfo`, then the public key shall have the following syntax:

```
SubjectPublicKeyInfo ::= SEQUENCE {
   algorithm         AlgorithmIdentifier {{ECPKAlgorithms}},
   subjectPublicKey  BIT STRING
}

AlgorithmIdentifier { ALGORITHM:IOSet } ::= SEQUENCE {
   algorithm   ALGORITHM.&id({IOSet}),
   parameters  ALGORITHM.&Type({IOSet}{@algorithm}) OPTIONAL
}

ECPKAlgorithms ALGORITHM ::= {
   ecPublicKeyAlgorithm,
   ...
}

ecPublicKeyAlgorithm ALGORITHM ::= { ECParameters IDENTIFIED BY id-
ecPublicKey }

ALGORITHM  ::= TYPE-IDENTIFIER

id-ecPublicKey OBJECT IDENTIFIER ::= { id-publicKeyType 1 }

id-publicKeyType OBJECT IDENTIFIER  ::= { ansi-X9-62 keyType (2) }
```

Note:

1. The object identifier `id-publicKeyType` represents the tree containing the object identifiers for each public key. It has the following value:

```
id-public-key-type OBJECT IDENTIFIER  ::= {ansi-X9.62 2}
```

2 The elliptic curve public key (an `ECPoint` which is an `OCTET STRING`) is mapped to a `subjectPublicKey` (a `BIT STRING`) as follows: the most significant bit of the `OCTET STRING` becomes the most significant bit of the `BIT STRING`, etc.; the least significant bit of the `OCTET STRING` becomes the least significant bit of the `BIT STRING`.

7. References

Elliptic curve cryptosystems were first proposed in 1985 independently by Neil Koblitz [22] and Victor Miller [30]. Since then, much research has been done towards improving the efficiency of these systems and evaluating their security. For a summary of this work, consult [28]. A description of a hardware implementation of an elliptic curve cryptosystem can be found in [7].

Three references on the theory of finite fields are the books of McEliece [27], Lidl and Niederreiter [26] and Jungnickel [20]. Lidl and Niederreiter's book [26] contains introductory material on polynomial and normal bases. The article [6] discusses methods which efficiently perform arithmetic operations in finite fields of characteristic 2. A hardware implementation of arithmetic in such fields which exploits the properties of optimal normal bases is described in [8].

The NIST Digital Signature Algorithm (DSA) is described in [4] and [33]. The Secure Hash Algorithm (SHA-1) is described in [5] and [32]. Abstract Syntax Notation One (ASN.1), Basic Encoding Rules (BER) and Distinguished Encoding Rules (DER) are described in [13], [16] [18]and [19] respectively.

1. ANSI X3.92-1981, *Data Encryption Algorithm*.
2. ANSI X9.9-1986 (revised), *Financial Institution Message Authentication* (*Wholesale*).
3. ANSI X9.17-1985, *Financial Institution Key Management* (*Wholesale*).
4. ANSI X9.30-1995, Part 1: *Public key cryptography using irreversible algorithms for the financial services industry: The Digital Signature Algorithm* (*Revised*).
5. ANSI X9.30-1993, Part 2: *Public key cryptography using irreversible algorithms for the financial services industry: The Secure Hash Algorithm 1* (*SHA-1*) (*Revised*).
6. G. AGNEW, T. BETH, R. MULLIN AND S. VANSTONE, *Arithmetic operations in GF*(2^m), Journal of Cryptology, **6** (1993), 3-13.
7. G. AGNEW, R. MULLIN AND S. VANSTONE, *An implementation of elliptic curve cryptosystems over* $F_{2^{155}}$, IEEE Journal on Selected Areas in Communications, **11** (1993), 804-813.
8. G. AGNEW, R. MULLIN, I. ONYSZCHUK AND S. VANSTONE, *An implementation for a fast public-key cryptosystem*, Journal of Cryptology, **3** (1991), 63-79.
9. M. BLAZE, W. DIFFIE, R. RIVEST, B. SCHNEIER, T. SHIMOMURA, E. THOMPSON, AND M. WIENER, *Minimal key lengths for symmetric ciphers to provide adequate commercial security*, January 1996.
10. E. BRICKELL, D. GORDON, K. MCCURLEY AND D. WILSON, *Fast Exponentiation with precomputation*, *Advances in Cryptology - EUROCRYPT* '92 Lecture Notes in Computer Science, **658** (1993), Springer-Verlag, 200-207.
11. T. ELGAMAL, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory, **31** (1985), 469-472.
12. IEEE P1363, *Standard for Public-Key Cryptography*, draft standard, 1997.
13. ITU-T Recommendation X.680 (1994), *Information Technology - Abstract Syntax Notation One* (*ASN.*1): *Specification of Basic Notation* (equivalent to ISO/IEC 8824-1:1995).

14. ITU-T Recommendation X.680 (1994), Amendment 1 (1995), *Information Technology - Abstract Syntax Notation One* (*ASN*.1): *Specification of Basic Notation - Amendment* 1: *Rules of Extensibility* (equivalent to ISO/IEC 8824-1: Amd.1: 1995).
15. ITU-T Recommendation X.680 (1994), Corr.1 (1995), *Information Technology - Abstract Syntax Notation One* (*ASN*.1): *Specification of Basic Notation - Technical Corrigendum* 1 (equivalent to ISO/IEC 8824-1: Corr.1: 1995).
16. ITU-T Recommendation X.681 (1994), *Information Technology - Abstract Syntax Notation One* (*ASN*.1): *Information Object Specification* (equivalent to ISO/IEC 8824-2: 1995).
17. ITU-T Recommendation X.681 (1994), Amendment 1 (1995), *Information Technology - Abstract Syntax Notation One* (*ASN*.1): *Information Object Specification - Amendment* 1: *Rules of Extensibility* (equivalent to ISO/IEC 8824-2: Amd.1: 1995).
18. ITU-T Recommendation X.682 (1994), *Information Technology - Abstract Syntax Notation One* (*ASN*.1): *Constraint Specification* (equivalent to ISO/IEC 8824-3: 1995).
19. ITU-T Recommendation X.690 (1994), *Information Technology - ASN*.1 Encoding Rules: *Specification of Basic Encoding Rules* (*BER*), *Canonical Encoding Rules* (*CER*) *and Distinguished Encoding Rules* (*DER*) (equivalent to ISO/IEC 8825-1: 1995).
20. D. JUNGNICKEL, *Finite Fields: Structure and Arithmetics*, B.I.-Wissenschaftsverlag, Mannheim, 1993.
21. D. KNUTH, *The Art of Computer Programming*, volume 2, 2nd edition, 1981.
22. N. KOBLITZ, *Elliptic curve cryptosystems*, Mathematics of Computation, **48** (1987), 203-209.
23. N. KOBLITZ, *A Course in Number Theory and Cryptography*, Springer-Verlag, 2nd edition, 1994.
24. R. LERCIER, *Finding good random elliptic curves for cryptosystems defined over* F_{2^n}, *Advances in Cryptography - EUROCRYPT '97*, Lecture Notes in Computer Science, **1233** (1997), Springer-Verlag, 379-392.
25. R. LERCIER AND F. MORAIN, *Counting the number of points on elliptic curves over finite fields: strategies and performances*, *Advances in Cryptology - EUROCRYPT* '95, Lecture Notes in Computer Science, **921** (1995), Springer-Verlag, 79-94.
26. R. LIDL AND H. NIEDERREITER, *Finite Fields*, Cambridge University Press, 1987.
27. R.J. MCELIECE, *Finite Fields for Computer Scientists and Engineers*, Kluwer Academic Publishers, 1987.
28. A. MENEZES, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.
29. A. MENEZES, T. OKAMOTO AND S. VANSTONE, *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Transactions on Information Theory, **39** (1993), 1639-1646.
30. V. MILLER, *Uses of elliptic curves in cryptography*, *Advances in Cryptology - CRYPTO '85*, Lecture Notes in Computer Science, **218** (1986), Springer-Verlag, 417-426.
31. R. MULLIN, I. ONYSZCHUK, S. VANSTONE AND R. WILSON, *Optimal normal bases in* $GF(p^n)$, Discrete Applied Mathematics, **22** (1988/89), 149-161.
32. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, *Secure Hash Standard* (*SHS*), FIPS Publication 180, May 1993.
33. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, *Digital Signature Standard*, FIPS Publication 186, 1993.
34. A. ODLYZKO, *The Future of Integer Factorization*, Cryptobytes, volume 1, number 2, summer 1995, 5-12.

35. P. VAN OORSCHOT AND M. WIENER, *Parallel Collision Search With Application To Hash Functions And Discrete Logarithms*, in Proceedings of the 2nd ACM Conference on Computer and Communications Security, Fairfax, Virginia, November 2-4, 1994, 210-218.

36. J. POLLARD, *Monte Carlo methods for index computation mod p*, Mathematics of Computation, **32** (1978), 918-924.

37. R. SCHOOF, *Elliptic curves over finite fields and the computation of square roots mod p*, Mathematics of Computation, **44** (1985), 483-494.

38. T. SATOH AND K. ARAKI, *Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves*, preprint, 1997.

39. N. SMART, Posting to sci.math.research, September 30 1977.

40. S. VAUDENAY, *Hidden collisions on DSS*, *Advances in Cryptology - CRYPTO* '96, Lecture Notes in Computer Science, **1109** (1996), Springer-Verlag, 83-88.

Appendix A An Overview of Elliptic Curve Systems
[Informative]

Many public-key cryptographic systems are based on exponentiation operations in large finite mathematical groups. The cryptographic strength of these systems is derived from the believed computational intractability of computing logarithms in these groups. The most common groups are the multiplicative groups of Z_p (the integers modulo a prime p) and F_{2^m} (characteristic 2 finite fields). The primary advantages of these groups are their rich theory, easily understood structure, and straightforward implementation. However, they are not the only groups that have the requisite properties. In particular, the mathematical structures known as elliptic curves have the requisite mathematical properties, a rich theory, and are especially amenable to efficient implementation in hardware or software.

The algebraic system defined on the points of an elliptic curve provides an alternate means to implement the ElGamal and ElGamal-like public-key encryption and signature protocols. These protocols are described in the literature in the algebraic system Z_p, the integers modulo p, where p is a prime. For example, the Digital Signature Algorithm (DSA) defined in X9.30-1-1994 is an ElGamal-like signature scheme defined over Z_p. The same protocol for signing can be defined over the points on an elliptic curve.

Elliptic curve systems as applied to ElGamal protocols were first proposed in 1985 independently by Neil Koblitz from the University of Washington, and Victor Miller, who was then at IBM, Yorktown Heights. The security of the cryptosystems using elliptic curves hinges on the intractability of the discrete logarithm problem in the algebraic system. Unlike the case of the discrete logarithm problem in finite fields, or the problem of factoring integers, there is no subexponential-time algorithm known for the elliptic curve discrete logarithm problem. The best algorithm known to date takes fully exponential time.

The primary advantage of elliptic curve systems is their apparent high cryptographic strength relative to the key size. The attractiveness of the elliptic curve cryptosystems may increase relative to other public-key cryptosystems, as computing power improvements force an increase in the key size. The significantly shorter key sizes may result in shorter certificates and system parameters. These potential advantages manifest themselves in many ways, including:

- storage efficiencies,
- bandwidth savings, and
- computational efficiencies.

The computational efficiencies may lead in turn to:

- higher speeds,
- lower power consumption, and
- code size reductions.

The computational efficiencies available with the use of these implementations, combined with the efficiencies of elliptic curves in general, are particularly beneficial in applications where bandwidth, processing capacity, power availability or storage are constrained. Such applications include wireless communications, handheld computing, broadcast communications and smart card applications.

Associated with any finite field F_q there are on the order of q different elliptic curves that can be formed and used for the cryptosystems. Thus, for a fixed finite field with q elements and with a large value of q, there are many choices for the elliptic curve group. Since each elliptic curve operation requires a number of more basic operations in the underlying finite field F_q, a finite field may be selected with a very efficient software or hardware implementation, and there remain an enormous number of choices for the elliptic curve.

This Standard describes the implementation of a signature algorithm which uses elliptic curves over a finite field F_q, where q is either a prime number or equal to 2^m for some positive integer m.

Appendix B The Elliptic Curve Analog of the DSA (ECDSA)
[Informative]

The elliptic curve algorithm (ECDSA) described in this Standard is the elliptic curve analog of a discrete logarithm algorithm that is usually described in the setting of F_p^* (also denoted Z_p^*), the multiplicative group of the integers modulo a prime. The following tables show the correspondence between the elements and operations of the group F_p^* and the elliptic curve group $E(F_q)$.

Table B-1: DSA and ECDSA Group Information

Group	F_p^*	$E(F_q)$
Group elements	The set of integers {1, 2, ..., p - 1}	Points (x,y) which satisfy the defining equation of the elliptic curve, plus the point at infinity $\mathcal{O}$.
Group operation	Multiplication modulo p	Addition of points
Notation	Elements: g, h Multiplication: $g \times h$ Exponentiation: g^a	Elements: P, Q Addition: $P + Q$ Multiple of a point (also called scalar multiplication): aP
Discrete logarithm problem	Given $g \in F_p^*$ and $h = g^a \bmod p$, find the integer a.	Given $P \in E(\mathbf{F_q})$ and $Q = aP$, find the integer a.

Table B-2: DSA and ECDSA Notation

DSA Notation	ECDSA Notation
q	n
g	P
x	d
y	Q

Table B-3: DSA and ECDSA Setup	
DSA Setup	ECDSA Setup
1. p and q are primes, q divides p - 1. 2. g is an element of order q in F_p*. 3. The group used is: $\{g^0, g^1, g^2, ..., g^{q-1}\}$.	1. E is an elliptic curve defined over the field $\boldsymbol{F}_q$. 2. P is a point of prime order n in $E(\boldsymbol{F}_q)$. 3. The group used is: $\{ \mathcal{O}, P, 2P, ..., (n-1)P\}$.

Table B-4: DSA and ECDSA Key Generation	
DSA Key Generation	ECDSA Key Generation
1. Select a random integer x in the interval $[1, q - 1]$. 2. Compute $y = g^x \bmod p$. 3. The private key is x. 4. The public key is y.	1. Select a statistically unique and unpredictable integer d in the interval $[1, n - 1]$. 2. Compute $Q = dP$. 3. The private key is d. 4. The public key is Q.

Table B-5: DSA and ECDSA Signature Generation	
DSA Signature Generation	ECDSA Signature Generation
1. Select a random integer k in the interval $[1, q - 1]$. 2. Compute $g^k \bmod p$. 3. Compute $r = (g^k \bmod p) \bmod q$. 4. Compute $e = H(M)$. 5. Compute $s = k^{-1}(e + xr) \bmod q$. 6. The signature for M is (r, s).	1. Select a statistically unique and unpredictable integer k in the interval $[1, n - 1]$. 2. Compute $kP = (x_1, y_1)$. 3. Compute $r = x_1 \bmod n$. 4. Compute $e = H(M)$. 5. Compute $s = k^{-1}(e + dr) \bmod n$. 6. The signature for M is (r, s).

Table B-6: DSA and ECDSA Signature Verification	
DSA Signature Verification	ECDSA Signature Verification
1. Compute $e = H(M)$. 2. Compute $s^{-1} \bmod q$. 3. Compute $u_1 = es^{-1} \bmod q$. 4. Compute $u_2 = rs^{-1} \bmod q$. 5. Compute $v' = g^{u_1} y^{u_2} \bmod p$. 6. Compute $v = v' \bmod q$. 7. Accept the signature if $v = r$.	1. Compute $e = H(M)$. 2. Compute $s^{-1} \bmod n$. 3. Compute $u_1 = es^{-1} \bmod n$. 4. Compute $u_2 = rs^{-1} \bmod n$. 5. Compute $u_1P + u_2Q = (x_1, y_1)$. 6. Compute $v = x_1 \bmod n$. 7. Accept the signature if $v = r$.

Appendix C Mathematical Background
[Informative]

C.1. The Finite Field F_p

Let p be a prime number. The finite field F_p is comprised of the set of integers

$\{0,1,2,..., p-1\}$

with the following arithmetic operations:

- *Addition*: If $a, b \in F_p$, then $a + b = r$, where r is the remainder when the integer $a + b$ is divided by p, $r \in [0, p\text{-}1]$. This is known as addition modulo p (mod p).
- *Multiplication*: If $a, b \in F_p$, then $ab = s$, where s is the remainder when the integer ab is divided by p, $s \in [0, p\text{-}1]$. This is known as multiplication modulo p (mod p).

Let F_p^* denote all the non-zero elements in F_p. In F_p, there exists at least one element g such that any non-zero element of F_p can be expressed as a power of g. Such an element g is called a *generator* (or *primitive element*) of F_p^*. That is

$F_p^* = \{g^i : 0 \le i \le p-2\}$.

The *multiplicative inverse* of $a = g^i \in F_p^*$, where $0 \le i \le p-2$, is:

$a^{-1} = g^{p-1-i}$.

Example 1: The finite field F_2

$F_2 = \{0, 1\}$. The addition and multiplication tables for F_2 are**:**

+	0	1
0	0	1
1	1	0

•	0	1
0	0	0
1	0	1

Example 2: The finite field F_{23}

$F_{23} = \{0,1,2,...,22\}$. Examples of the arithmetic operations in F_{23} are:

$12 + 20 = 32 \bmod 23 = 9$, since the remainder is 9 when 32 is divided by 23.
$8 \cdot 9 = 72 \bmod 23 = 3$, since the remainder is 3 when 72 is divided by 23.

The element 5 is a generator of F_{23}^*. The powers of 5 modulo 23 are:

$5^0 = 1$	$5^1 = 5$	$5^2 = 2$	$5^3 = 10$	$5^4 = 4$	$5^5 = 20$
$5^6 = 8$	$5^7 = 17$	$5^8 = 16$	$5^9 = 11$	$5^{10} = 9$	$5^{11} = 22$
$5^{12} = 18$	$5^{13} = 21$	$5^{14} = 13$	$5^{15} = 19$	$5^{16} = 3$	$5^{17} = 15$
$5^{18} = 6$	$5^{19} = 7$	$5^{20} = 12$	$5^{21} = 14$	$5^{22} = 1$.	

C.2. The Finite Field F_{2^m}

There are many ways to construct a finite field with 2^m elements. The field F_{2^m} can be viewed as a vector space of dimension m over F_2. That is, there exists m elements $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ in F_{2^m} such that each element $\alpha \in F_{2^m}$ can be uniquely written in the form:

$\alpha = a_0\alpha_0 + a_1\alpha_1 + \ldots + a_{m-1}\alpha_{m-1}$, where $a_i \in \{0,1\}$.

Such a set $\{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$ of elements is called a *basis* of F_{2^m} over F_2. Given such a basis, we can represent a field element α as the binary vector $(a_0, a_1, \ldots, a_{m-1})$. Addition of field elements is performed by bitwise XOR-ing the vector representations.

There are many different bases of F_{2^m} over F_2. Some bases lead to more efficient software and/or hardware implementations of the arithmetic in F_{2^m} than other bases. In this section, three kinds of bases are discussed. Section C.2.1 introduces *polynomial bases* which use polynomial addition, multiplication, division and

remainder. Section C.2.2 introduces special kinds of polynomial bases called *trinomial* and *pentanomial bases*. Section C.2.3 introduces *normal bases*. Section C.2.4 introduces special kinds of normal bases called *optimal normal bases* (ONB).

C.2.1. Polynomial Bases

Let $f(x)= x^m+ f_{m-1}x^{m-1}+ \ldots + f_2x^2+ f_1x+ f_0$ (where $f_i \in F_2$ for i = 0, ..., m-1) be an irreducible polynomial of degree m over F_2, i.e., $f(x)$ cannot be factored into two polynomials over F_2, each of degree less than m. $f(x)$ is called the *reduction polynomial*. The *finite field* F_{2^m} is comprised of all polynomials over F_2 of degree less than m:

$F_{2^m} = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_1x + a_0 : a_i \in \{0,1\}\}$.

The field element $(a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_1x + a_0)$ is usually denoted by the binary string $(a_{m-1}...a_1a_0)$ of length m, so that

$F_{2^m} = \{(a_{m-1}... a_1a_0): a_i \in \{0,1\}\}$.

Thus the elements of F_{2^m} can be represented by the set of all binary strings of length m. The multiplicative identity element (1) is represented by the bit string (00…01), while the zero element is represented by the bit string of all 0's.

Field elements are added and multiplied as follows:

C.2.1.a. Field addition

$(a_{m-1}...a_1a_0) + (b_{m-1}\ldots b_1b_0) = (c_{m-1}\ldots c_1c_0)$
where $c_i = a_i \oplus b_i$. That is, field addition is performed componentwise.

C.2.1.b. Field multiplication

$(a_{m-1}...a_1a_0) \bullet (b_{m-1}\ldots b_1b_0) = (r_{m-1}\ldots r_1r_0)$,
where the polynomial $(r_{m-1}x^{m-1}+\ldots+ r_1x+ r_0)$ is the remainder when the polynomial
$(a_{m-1}x^{m-1} +\ldots + a_1x + a_0) \bullet (b_{m-1}x^{m-1} +\ldots + b_1x + b_0)$
is divided by $f(x)$ over F_2.

This method of representing F_{2^m} is called a *polynomial basis representation*, and $\{1, x, x^2, \ldots, x^{m-1}\}$ is called a *polynomial basis* of F_{2^m} over F_2.

Note that F_{2^m} contains exactly 2^m elements. Let $F^*_{2^m}$ denote the set of all non-zero elements in F_{2^m}. There exists at least one element g in F_{2^m} such that any non-zero element of F_{2^m} can be expressed as a power of g. Such an element g is called a *generator* (or *primitive element*) of F_{2^m}. That is

$F^*_{2^m} = \{g^i : 0 \le i \le 2^m - 2\}$.

The *multiplicative inverse* of $a = g^i \in F^*_{2^m}$, where $0 \le i \le 2^m - 2$, is:

$a^{-1} = g^{2^m-1-i}$.

Example 3: The finite field F_{2^4} using a polynomial basis representation

Take $f(x) = x^4 + x + 1$ over F_2; it can be verified that $f(x)$ is irreducible over F_2. Then the elements of F_{2^4} are:

(0000) (1000) (0100) (1100) (0010) (1010) (0110) (1110)
(0001) (1001) (0101) (1101) (0011) (1011) (0111) (1111).

As examples of field arithmetic, we have:
(1101) + (1001) = (0100), and
(1101) • (1001) = (1111) since

$$\begin{aligned}(x^3 + x^2 + 1)(x^3+ 1) &= x^6 + x^5 + x^2 + 1\\ &= (x^4+ x +1)(x^2 + x) + (x^3 + x^2 + x +1)\\ &= x^3 + x^2 + x +1 \bmod f(x).\end{aligned}$$

i.e., $x^3 + x^2 + x +1$ is the remainder when $(x^3 + x^2 +1) \bullet (x^3 +1)$ is divided by $f(x)$.
The multiplicative identity is (0001).

$F^*_{2^4}$ can be generated by one element, $\alpha = x$. The powers of α are:

$\alpha^0 = (0001)$ $\alpha^1 = (0010)$ $\alpha^2 = (0100)$ $\alpha^3 = (1000)$
$\alpha^4 = (0011)$ $\alpha^5 = (0110)$ $\alpha^6 = (1100)$ $\alpha^7 = (1011)$
$\alpha^8 = (0101)$ $\alpha^9 = (1010)$ $\alpha^{10} = (0111)$ $\alpha^{11} = (1110)$
$\alpha^{12} = (1111)$ $\alpha^{13} = (1101)$ $\alpha^{14} = (1001)$.

C.2.2. Trinomial and Pentanomial Bases

A *trinomial basis* (TPB) and a *pentanomial basis* (PPB) are special types of polynomial bases. A *trinomial* over F_2 is a polynomial of the form $x^m + x^k + 1$, where $1 \le k \le m$ -1. A *pentanomial* over F_2 is a polynomial of the form $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, where $1 \le k_1 < k_2 < k_3 \le m$ -1.

A *trinomial basis representation* of F_{2^m} is a polynomial basis representation determined by an irreducible trinomial $f(x) = x^m + x^k + 1$ of degree m over F_2. Such trinomials only exist for certain values of m. Example 3 above is an example of a trinomial basis representation of the finite field F_{2^4}.

A *pentanomial basis* representation of F_{2^m} is a polynomial basis representation determined by an irreducible pentanomial $f(x) = x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, of degree m over F_2. Such pentanomials exist for all values of $m \ge 4$.

C.2.3. Normal Bases

A *normal basis* of F_{2^m} over F_2 is a basis of the form:
$\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\}$,
where $\beta \in F_{2^m}$.

Such a basis always exists. Given any element $\alpha \in F_{2^m}$, we can write $\alpha = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, where $a_i \in \{0,1\}$.

This field element α is denoted by the binary string $(a_0 a_1 a_2 \ldots a_{m-1})$ of length m, so that $F_{2^m} = \{(a_0 a_1 \ldots a_{m-1}) : a_i \in \{0,1\}\}$.

Note that, by convention, the ordering of bits is different from that of a polynomial basis representation (Section C.2.1).

The multiplicative identity element (1) is represented by the bit string of all 1's (11. . .11), while the zero element is represented by the bit string of all 0's.

Since squaring is a linear operator in F_{2^m}, we have:

$$\alpha^2 = \sum_{i=0}^{m-1} a_i^2 (\beta^{2^i})^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}} = \sum_{i=0}^{m-1} a_{i-1} \beta^{2^i} = (a_{m-1} a_0 \ldots a_{m-2}),$$

with indices reduced modulo m. Hence a normal basis representation of F_{2^m} is advantageous because squaring a field element can then be accomplished by a simple rotation of the vector representation, an operation that is easily implemented in hardware.

C.2.4. Optimal Normal Bases

In Example 3, the field F_{2^4} was described using polynomial multiplication, division and remainders. An optimal normal basis representation, as defined in Section 4.1.4, may also be used to construct the field F_{2^4}. Note that a Type I ONB does indeed exist for F_{2^4}.

Example 4: The finite field F_{2^4} using an optimal normal basis representation

As in Example 3, the elements of F_{2^4} are the binary 4-tuples:

(0000) (0001) (0010) (0011) (0100) (0101) (0110) (0111)
(1000) (1001) (1010) (1011) (1100) (1101) (1110) (1111).

Field elements are added and multiplied as follows:

C.2.4.a. Field addition:

$(a_0a_1a_2a_3) + (b_0b_1b_2b_3) = (c_0c_1c_2c_3)$
where $c_i = a_i \oplus b_i$.
In other words, field addition is performed by simply XORing the vector representation.

C.2.4.b. Field multiplication:

The setup for multiplication is done as follows. See Section 4.1.4 for a description of the steps that are performed.

1. $f(x) = x^4 + x^3 + x^2 + x + 1$.
2. Since $x \bmod f(x) = x$,
   $x^2 \bmod f(x) = x^2$,
   $x^4 \bmod f(x) = x^3 + x^2 + x + 1$,
   $x^8 \bmod f(x) = x^3$,

   the matrix A is computed as:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

3. The inverse of A is:

$$A^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

4. The matrix T is computed as:

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

5. The λ_{ij} terms which are 1 are: λ_{02}, λ_{12}, λ_{13}, λ_{20}, λ_{21}, λ_{31} and λ_{33}.

   For this particular example, multiplication is defined to be:
   $(a_0a_1a_2a_3) \bullet (b_0b_1b_2b_3) = (c_0c_1c_2c_3)$, where
   $c_0 = a_0b_2 \oplus a_1(b_2 \oplus b_3) \oplus a_2(b_0 \oplus b_1) \oplus a_3(b_1 \oplus b_3)$
   $c_1 = a_1b_3 \oplus a_2(b_3 \oplus b_0) \oplus a_3(b_1 \oplus b_2) \oplus a_0(b_2 \oplus b_0)$
   $c_2 = a_2b_0 \oplus a_3(b_0 \oplus b_1) \oplus a_0(b_2 \oplus b_3) \oplus a_1(b_3 \oplus b_1)$
   $c_3 = a_3b_1 \oplus a_0(b_1 \oplus b_2) \oplus a_1(b_3 \oplus b_0) \oplus a_2(b_0 \oplus b_2)$.

With these definitions of addition and multiplication, the 16 binary 4-tuples form a field.

Notes:

1. When an ONB is used, the multiplicative identity in F_{2^4} is (1111). (In Example 3, when a polynomial basis was used, it was (0001).)
2. Referring to the equations in step 5 above, note that:

   $(a_0a_1a_2a_3) \bullet (a_0a_1a_2a_3) = (a_0a_1a_2a_3)^2 = (a_3a_0a_1a_2)$,

so the square of a field element is simply a *right cyclic shift* of its vector representation.

3. The formula for c_1 in the multiplication can be obtained by adding a 1 to each subscript in the formula for c_0 (where the subscripts are reduced modulo 4). The formula for c_2 can be obtained by adding 2 to each subscript in the formula for c_0 and reducing the subscripts modulo 4. The formula for c_3 can likewise be obtained.

4. From the preceding remark we see that a circuit to compute c_0 from $(a_0a_1a_2a_3)$ and $(b_0b_1b_2b_3)$ can be used to compute c_1 by applying $(a_1a_2a_3a_0)$ and $(b_1b_2b_3b_0)$ to the circuit. Similarly, c_2 and c_3 can be computed by shifting the input vectors to the left.

As an example of how to compute in this representation of F_{2^4}, we have

$(1001) \cdot (1101) = (c_0c_1c_2c_3)$

where:

$c_0 = (1)(0) \oplus (0)(0 \oplus 1) \oplus (0)(1 \oplus 1) \oplus (1)(1 \oplus 1) = 0$

$c_1 = (0)(1) \oplus (0)(1 \oplus 1) \oplus (1)(1 \oplus 0) \oplus (1)(0 \oplus 1) = 0$

$c_2 = (0)(1) \oplus (1)(1 \oplus 1) \oplus (1)(0 \oplus 1) \oplus (0)(1 \oplus 1) = 1$

$c_3 = (1)(1) \oplus (1)(1 \oplus 0) \oplus (0)(1 \oplus 1) \oplus (0)(1 \oplus 0) = 0.$

That is,

$(1001) \cdot (1101) = (0010).$

Also,

$$(1010)^{10} = (1010)^2 \cdot (1010)^8$$
$$= (0101) \cdot (0101)$$
$$= (1010),$$

and

$$(1101)^9 = (1101) \cdot (1101)^8$$
$$= (1101) \cdot (1011)$$
$$= (0001).$$

In this representation, $F^*_{2^4}$ can be generated by the powers of $\alpha = (1100)$:

$\alpha^0 = (1111)$ $\alpha^1 = (1100)$ $\alpha^2 = (0110)$ $\alpha^3 = (0100)$
$\alpha^4 = (0011)$ $\alpha^5 = (1010)$ $\alpha^6 = (0010)$ $\alpha^7 = (0111)$
$\alpha^8 = (1001)$ $\alpha^9 = (1000)$ $\alpha^{10} = (0101)$ $\alpha^{11} = (1110)$
$\alpha^{12} = (0001)$ $\alpha^{13} = (1101)$ $\alpha^{14} = (1011).$

This method of representing F_{2^4} is called an *optimal normal basis representation*.

C.3. Elliptic Curves over F_p

Let $p > 3$ be a prime number. Let $a, b \in F_p$ be such that $4a^3 + 27b^2 \neq 0$ in F_p. An *elliptic curve* $E(F_p)$ over F_p defined by the parameters a and b is the set of solutions (x, y), $x, y \in F_p$, to the equation: $y^2 = x^3 + ax +$

b, together with an extra point $\mathcal{O}$, the *point at infinity*. The number of points in $E(F_p)$ is denoted by $\#E(F_p)$. The Hasse Theorem tells us that

$$p + 1 - 2\sqrt{p} \le \#E(F_p) \le p + 1 + 2\sqrt{p}.$$

The set of points $E(F_p)$ forms a group with the following addition rules:

(i) $\mathcal{O} + \mathcal{O} = \mathcal{O}$.

(ii) $(x, y) + \mathcal{O} = \mathcal{O} + (x, y) = (x, y)$ for all $(x, y) \in E(F_p)$.

(iii) $(x, y) + (x,-y) = \mathcal{O}$ for all $(x, y) \in E(F_p)$ (i.e., the negative of the point (x, y) is $-(x, y) = (x,-y)$).

(iv) (Rule for adding two distinct points that are not inverses of each other)

Let: $(x_1,y_1) \in E(F_p)$ and $(x_2,y_2) \in E(F_p)$ be two points such that $x_1 \neq x_2$.
Then $(x_1,y_1) + (x_2,y_2) = (x_3,y_3)$, where:

$$x_3 = \lambda^2 - x_1 - x_2,\ y_3 = \lambda (x_1 - x_3) - y_1 \text{ and } \lambda = \frac{y_2 - y_1}{x_2 - x_1}.$$

(v) (Rule for doubling a point)

Let $(x_1, y_1) \in E(F_p)$ be a point with $y_1 \neq 0$.
Then $2(x_1, y_1) = (x_3, y_3)$, where:

$$x_3 = \lambda^2 - 2x_1,\ y_3 = \lambda (x_1 - x_3) - y_1, \text{ and } \lambda = \frac{3x_1^2 + a}{2y_1}.$$

The group $E(F_p)$ is *abelian*, which means that $P_1 + P_2 = P_2 + P_1$ for all points P_1 and P_2 in $E(F_p)$. The curve is said to be *supersingular* if $\#E(F_p) = p + 1$; otherwise it is *non-supersingular*. Only non-supersingular curves shall be in compliance with this standard (see Appendix D).

Example 5: An elliptic curve over F_{23}

Let $y^2 = x^3 + x + 1$ be an equation over F_{23}. Here $a = 1$ and $b = 1$. Then the solutions over F_{23} to the equation of the elliptic curve are:

(0,1)	(0,22)	(1,7)	(1,16)	(3,10)	(3,13)	(4,0)	(5,4)	(5,19)
(6,4)	(6,19)	(7,11)	(7,12)	(9,7)	(9,16)	(11,3)	(11,20)	(12,4)
(12,19)	(13,7)	(13,16)	(17,3)	(17,20)	(18,3)	(18,20)	(19,5)	(19,18).

The solutions were obtained by trial and error. The group $E(F_{23})$ has 28 points (including the point at infinity $\mathcal{O}$). The following are examples of the group operation.

1. Let $P_1 = (3,10)$, $P_2 = (9,7)$, $P_1 + P_2 = (x_3, y_3)$. Compute

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \frac{7-10}{9-3} = \frac{-3}{6} = \frac{-1}{2} = 11 \in F_{23},$$

$x_3 = \lambda^2 - x_1 - x_2 = 11^2 - 3 - 9 = 6 - 3 - 9 = -6 = 17$,
$y_3 = \lambda(x_1 - x_3) - y_1 = 11(3 - 17) - 10 = 11(9) - 10 = 89 = 20$.
Therefore $P_1 + P_2 = (17, 20)$.

2. Let $P_1 = (3, 10)$, $2P_1 = (x_3, y_3)$. Compute

$$\lambda = \frac{3x_1^2 + a}{2y_1} = \frac{3(3^2)+1}{20} = \frac{5}{20} = \frac{1}{4} = 6,$$

$x_3 = \lambda^2 - 2x_1 = 6^2 - 6 = 30 = 7$,
$y_3 = \lambda(x_1 - x_3) - y_1 = 6(3 - 7) - 10 = -24 - 10 = -11 = 12$.
Therefore $2P_1 = (7, 12)$.

C.4. Elliptic Curves over F_{2^m}

A non-supersingular *elliptic curve* $E(F_{2^m})$ over F_{2^m} defined by the parameters $a, b \in F_{2^m}$, $b \neq 0$, is the set of solutions (x, y), $x \in F_{2^m}$, $y \in F_{2^m}$, to the equation $y^2 + xy = x^3 + ax^2 + b$ together with an extra point $\mathcal{O}$, the *point at infinity*. The number of points in $E(F_{2^m})$ is denoted by $\#E(F_{2^m})$. The Hasse Theorem tells us that

$$q + 1 - 2\sqrt{q} \leq \#E(F_{2^m}) \leq q + 1 + 2\sqrt{q},$$

where $q = 2^m$. Furthermore, $\#E(F_{2^m})$ is even.

The set of points $E(F_{2^m})$ forms a group with the following addition rules:

(i) $\mathcal{O} + \mathcal{O} = \mathcal{O}$.

(ii) $(x, y) + \mathcal{O} = \mathcal{O} + (x, y) = (x, y)$ for all $(x, y) \in E(F_{2^m})$.

(iii) $(x, y) + (x, x + y) = \mathcal{O}$ for all $(x, y) \in E(F_{2^m})$ (i.e., the negative of the point (x, y) is $-(x, y) = (x, x + y)$).

(iv) (Rule for adding two distinct points that are not inverses of each other)

Let $(x_1, y_1) \in E(F_{2^m})$ and $(x_2, y_2) \in E(F_{2^m})$ be two points such that $x_1 \neq x_2$. Then $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, where:

$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$, $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$, and $\lambda = \dfrac{y_1 + y_2}{x_1 + x_2}$.

(v) (Rule for doubling a point)

Let $(x_1, y_1) \in E(F_{2^m})$ be a point with $x_1 \neq 0$. Then $2(x_1, y_1) = (x_3, y_3)$, where

$x_3 = \lambda^2 + \lambda + a$, $y_3 = x_1^2 + (\lambda + 1)\, x_3$, and $\lambda = x_1 + \dfrac{y_1}{x_1}$.

The group $E(F_{2^m})$ is *abelian*, which means that $P_1 + P_2 = P_2 + P_1$ for all points P_1 and P_2 in $E(F_{2^m})$.

We now give two examples of elliptic curves over F_{2^4}. Example 6 uses a trinomial basis representation for the field, and Example 7 uses an optimal normal basis representation.

Example 6: An elliptic curve over F_{2^4}. A trinomial basis representation is used for the elements of F_{2^4}.

Consider the field F_{2^4} generated by the root $\alpha = x$ of the irreducible polynomial:

$$f(x) = x^4 + x + 1.$$

(See Example 3.) The powers of α are:

$\alpha^0 = (0001)$	$\alpha^1 = (0010)$	$\alpha^2 = (0100)$	$\alpha^3 = (1000)$
$\alpha^4 = (0011)$	$\alpha^5 = (0110)$	$\alpha^6 = (1100)$	$\alpha^7 = (1011)$
$\alpha^8 = (0101)$	$\alpha^9 = (1010)$	$\alpha^{10} = (0111)$	$\alpha^{11} = (1110)$
$\alpha^{12} = (1111)$	$\alpha^{13} = (1101)$	$\alpha^{14} = (1001)$	$\alpha^{15} = \alpha^0 = (0001)$.

Consider the non-supersingular elliptic curve over F_{2^4} with defining equation:

$$y^2 + xy = x^3 + \alpha^4 x^2 + 1.$$

Here, $a = \alpha^4$ and $b = 1$. The notation for this equation can be expressed as follows, since the multiplicative identity is (0001):

$$(0001)\, y^2 + (0001)\, xy = (0001)\, x^3 + (0011)\, x^2 + (0001).$$

Then the solutions over F_{2^4} to the equation of the elliptic curve are:

$(0, 1)$ $(1, \alpha^6)$ $(1, \alpha^{13})$ (α^3, α^8) (α^3, α^{13}) (α^5, α^3) (α^5, α^{11})
(α^6, α^8) (α^6, α^{14}) (α^9, α^{10}) (α^9, α^{13}) (α^{10}, α^1) (α^{10}, α^8) $(\alpha^{12}, 0)$ $(\alpha^{12}, \alpha^{12})$.

The group $E(F_{2^4})$ has 16 points (including the point at infinity $\mathcal{O}$). The following are examples of the group operation.

1. Let $P_1 = (x_1, y_1) = (\alpha^6, \alpha^8)$, $P_2 = (x_2, y_2) = (\alpha^3, \alpha^{13})$, and $P_1 + P_2 = (x_3, y_3)$. Then:

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} = \frac{\alpha^8 + \alpha^{13}}{\alpha^6 + \alpha^3} = \alpha,$$
$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a = \alpha^2 + \alpha + \alpha^6 + \alpha^3 + \alpha^4 = 1,$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 = \alpha(\alpha^6 + 1) + 1 + \alpha^8 = \alpha^{13}.$$

2. If $2P_1 = (x_3, y_3)$, then:

$$\lambda = x_1 + \frac{y_1}{x_1} = \alpha^6 + \frac{\alpha^8}{\alpha^6} = \alpha^3,$$
$$x_3 = \lambda^2 + \lambda + a = \alpha^6 + \alpha^3 + \alpha^4 = \alpha^{10},$$
$$y_3 = x_1^2 + (\lambda + 1)x_3 = \alpha^{12} + (\alpha^3 + 1)\alpha^{10} = \alpha^8.$$

Example 7: An elliptic curve over F_{2^4}. An optimal normal basis representation is used for the elements of F_{2^4}.

Consider the field F_{2^4} given by the optimal normal basis representation in Example 4. That is, the elements of F_{2^4} are the set of all binary 4-tuples with multiplication given by the formulae in Section 4.1.4.b. Recall that $\alpha = (1100)$ is a generator for the non-zero elements, and (1111) is the multiplicative identity. The powers of α are:

$\alpha^0 = (1111)$ $\alpha^1 = (1100)$ $\alpha^2 = (0110)$ $\alpha^3 = (0100)$
$\alpha^4 = (0011)$ $\alpha^5 = (1010)$ $\alpha^6 = (0010)$ $\alpha^7 = (0111)$
$\alpha^8 = (1001)$ $\alpha^9 = (1000)$ $\alpha^{10} = (0101)$ $\alpha^{11} = (1110)$
$\alpha^{12} = (0001)$ $\alpha^{13} = (1101)$ $\alpha^{14} = (1011)$ $\alpha^{15} = \alpha^0 = (1111)$.

Consider the non-supersingular curve over F_{2^4} defined by the equation

$$E : y^2 + xy = x^3 + \alpha^3.$$

Here, $a = 0$ and $b = \alpha^3$. The notation for this equation can be expressed as follows since the multiplicative identity is (1111):

$$(1111)\, y^2 + (1111)\, xy = (1111)\, x^3 + (0100).$$

The solutions over F_{2^4} to the elliptic curve equation are:

$(0, \alpha^9)$ $(\alpha, 0)$ (α, α) (α^3, α^5) (α^3, α^{11}) (α^4, α^3) (α^4, α^7)
(α^5, α^3) (α^5, α^{11}) $(\alpha^6, 0)$ (α^6, α^6) (α^8, α^3) (α^8, α^{13})
$(\alpha^{11}, 0)$ $(\alpha^{11}, \alpha^{11})$ (α^{12}, α^8) (α^{12}, α^9) (α^{13}, α^2) $(\alpha^{13}, \alpha^{14})$.

Since there are 19 solutions to the equation in F_{2^4}, the group $E(F_{2^4})$ has 19 + 1 = 20 elements (including the point at infinity). This group turns out to be a cyclic group of order 20. If we take $G = (\alpha^3, \alpha^5)$ and use the addition formulae, we find that:

$1G = (\alpha^3, \alpha^5)$	$2\ G = (\alpha^4, \alpha^3)$	$3\ G = (\alpha^{13}, \alpha^2)$	$4\ G = (\alpha, 0)$	$5\ G = (\alpha^{12}, \alpha^8)$
$6\ G = (\alpha^8, \alpha^3)$	$7\ G = (\alpha^{11}, 0)$	$8\ G = (\alpha^5, \alpha^{11})$	$9\ G = (\alpha^6, 0)$	$10\ G = (0, \alpha^9)$
$11\ G = (\alpha^6, \alpha^6)$	$12\ G = (\alpha^5, \alpha^3)$	$13\ G = (\alpha^{11}, \alpha^{11})$	$14\ G = (\alpha^8, \alpha^{13})$	$15\ G = (\alpha^{12}, \alpha^9)$
$16\ G = (\alpha,\alpha)$	$17\ G = (\alpha^{13}, \alpha^{14})$	$18\ G = (\alpha^4, \alpha^7)$	$19\ G = (\alpha^3, \alpha^{11})$	$20\ G = \mathcal{O}$.

Appendix D Security Considerations
[Informative]

This appendix is provided as initial guidance for implementers of this Standard. This information should be expected to change over time. Implementers should review the current state-of-the-art in attacks on elliptic curve systems at the time of implementation.

The best attacks known on the elliptic curve discrete logarithm problem, which is the basis for the security of elliptic curve systems, are summarized. Estimates of security levels for elliptic curve parameters of various sizes are provided.

Notation

E denotes an elliptic curve over the finite field F_q. $P \in E(F_q)$ is a point of order n, where n is a prime number and $n > 2^{160}$.

D.1. The Elliptic Curve Discrete Logarithm Problem

The elliptic curve discrete logarithm problem (ECDLP) is the following: given E, P and $Q \in E$, determine the integer l, $0 \leq l \leq n-1$, such that $Q = lP$, provided that such an integer exists.

The best general algorithm known to date for ECDLP is the Pollard-ρ method [36] which takes about $\sqrt{\pi n / 2}$ steps, where each step is an elliptic curve addition. The Pollard-ρ method can be parallelized (see [35]) so that if m processors are used, then the expected number of steps by each processor before a single discrete logarithm is obtained is $(\sqrt{\pi n / 2}) / m$.

The special classes of elliptic curves, called *supersingular curves*, have been prohibited in this Standard by the requirement of the MOV condition (see Section F.1.1). These curves have been prohibited because there is a method for efficiently reducing the discrete logarithm problem in these curves to the discrete logarithm problem in a finite field.

Also, the special class of elliptic curves called F_q*-anomalous curves* have been prohibited by the requirement of the Anomalous condition (see Section F.1.2) because there is an efficient algorithm for computing discrete logarithms in $E(F_q)$ where E is an anomalous curve over F_q (i.e. $\#E(F_q) = q$).

To guard against existing attacks on ECDLP, one should select an elliptic curve E over F_q such that:

1. The order $\#E(F_q)$ is divisible by a large prime $n > 2^{160}$;
2. The MOV condition (Section F.1.1) holds; and
3. The Anomalous condition (Section F.1.2) holds.

Furthermore, to guard against possible future attacks against special classes of non-supersingular curves, it is prudent to select an elliptic curve at random. Section F.3.3 describes a method for selecting an elliptic curve *verifiably* at random.

D.2. Software Attacks

Assume that a 1 MIPS (Million Instructions Per Second) machine can perform 4×10^4 elliptic curve additions per second. (This estimate is indeed high — an ASIC (Application Specific Integrated Circuit) built for performing elliptic curve operations over the field $F_{2^{155}}$ has a 40 MHz clock-rate and can perform roughly 40,000 elliptic additions per second.) Then, the number of elliptic curve additions that can be performed by a 1 MIPS machine in one year is

$$(4 \times 10^4) \cdot (60 \times 60 \times 24 \times 365) \approx 2^{40}.$$

Table D-1 shows the computing power required to compute a single discrete logarithm for various values of *n*. As an example, if 10,000 computers each rated at 1,000 MIPS are available, and $n \approx 2^{160}$, then an elliptic curve discrete logarithm can be computed in 96,000 years.
Odlyzko [34] has estimated that if 0.1% of the world's computing power were available for one year to work on a collaborative effort to break some challenge cipher, then the computing power available would be 10^8 MIPS years in 2004 and 10^{10} to 10^{11} MIPS years in 2014.

Field size (in bits)	Size of *n* (in bits)	$\sqrt{\pi\, n / 2}$	MIPS years
163	160	2^{80}	9.6×10^{11}
191	186	2^{93}	7.9×10^{15}
239	234	2^{117}	1.3×10^{23}
359	354	2^{177}	1.5×10^{41}
431	426	2^{213}	1.0×10^{52}

Table D-1: Computing power required to compute elliptic curve logarithms with the Pollard-ρ method.

To put the numbers in Table D-1 into some perspective, Table D-2 (due to Odlyzko [34]) shows the computing power required to factor integers with 1995 versions of the general number field sieve.

Size of integer to be factored (in bits)	MIPS years
512	3×10^4
768	2×10^8
1024	3×10^{11}
1280	1×10^{14}
1536	3×10^{16}
2048	3×10^{20}

Table D-2: Computing power required to factor integers using the general number field sieve.

D.3. Hardware Attacks

A more promising attack (for well-funded attackers) on elliptic curve systems would be to build special-purpose hardware for a parallel search. Van Oorschot and Wiener [35] provide a detailed study of such a possibility. In their 1994 study, they estimated that if $n \approx 10^{36} \approx 2^{120}$, then a machine with m = 325,000 processors that could be built for about $10 million would compute a single discrete logarithm in about 35 days.
It must be emphasized that these estimates were made for specific elliptic curve parameters having $n \approx 10^{36} \approx 2^{120}$. This Standard mandates that the parameter *n* should satisfy

$$n > 2^{160} \approx 10^{48},$$

and hence the hardware attacks are infeasible.

D.4. Key Length Considerations

It should be noted that for the software and hardware attacks described above, the computation of a single elliptic curve discrete logarithm has the effect of revealing a *single* user's private key. The same effort must be repeated in order to determine another user's private key.
In [9], Blaze et al. report on the minimum key lengths required for secure symmetric-key encryption schemes (such as DES and IDEA). Their report provides the following conclusion:

To provide adequate protection against the most serious threats — well-funded commercial enterprises or government intelligence agencies — keys used to protect data today should be at least 75 bits long. To protect information adequately for the next 20 years in the face of expected advances in computing power, keys in newly-deployed systems should be at least 90 bits long.

Extrapolating these conclusions to the case of elliptic curves, we see that n should be at least 150 bits for short-term security, and at least 180 bits for medium-term security. This extrapolation is justified by the following considerations:

1. Exhaustive search through a k-bit symmetric-key cipher takes about the same time as the Pollard-ρ algorithm applied to an elliptic curve having a $2k$-bit parameter n.
2. Both exhaustive search with a symmetric-key cipher and the Pollard-ρ algorithm can be parallelized with a linear speedup.
3. A basic operation with elliptic curves (addition of two points) is computationally more expensive than a basic operation in a symmetric-key cipher (encryption of one block).
4. In both symmetric-key ciphers and elliptic curve systems, a "break" has the same effect: it recovers a single private key.

D.5. Vaudenay's Attack

Vaudenay [40] presented some attacks on the DSA where an adversary can forge one signature if she can select the system parameters. One attack relies on the fact that the DSA signature hash function is actually SHA-1 mod q, not merely SHA-1. In ECDSA, Vaudenay's attack is thwarted because $n > 2^{160}$.

D.6. System Parameter Generation Decisions

There are several decisions to make during system parameter generation (besides the key size) that affects security. These are as follows:

1. Whether to use a probabilistic or deterministic primality test. If using a probabilistic test, what number of individual tests T to be used. The Standard specifies that when using a probabilistic test, $T \geq 50$. If $T = 50$ then the chance that a probable prime output by this test is actually composite is less than 2^{-100}. This is a very remote possibility. If additional assurance is desired, a larger value for T may be specified or a deterministic primality test run.
2. Whether to use the canonical seeded hash (Section F.3.3) to determine the elliptic curve parameters (a and b). For the DSA, there is the possibility that a particularly poor choice of system parameters could lead to an attack. To address this, the DSA requires the use of a canonical seeded hash to generate the system parameters p and q, as this provides an assurance that p and q were generated arbitrarily. The analogous attack on the ECDSA does not apply, and there are no known poor choices for system parameters that are not already excluded by this Standard. The use of a specific elliptic curve may allow performance improvements over the use of an arbitrary elliptic curve. For these reasons, this Standard allows both the choice of a particular elliptic curve or the generation of an arbitrary curve through the use of a canonical seeded hash function. One may desire to use an arbitrary curve when security considerations are so preeminent that the possible performance impact is not a factor in the decision.
3. How large the MOV threshold B (see Section F.1) should be. The MOV threshold B is a positive integer B such that taking discrete logarithms over F_{q^B} is at least as difficult as taking elliptic curve discrete logarithms over F_q. For this Standard, $B \geq 20$. For example, for an elliptic curve over $F_{2^{191}}$, this means that all elliptic curves that are able to be mapped into finite fields with an order up to around 2^{3800} are eliminated from consideration. The value $B = 20$ is a conservative choice, and is sufficient to ensure resistance against the reduction attack.
4. What values to use for l_{max} and r_{min} when determining n, the order of the distinguished point P (see Section F.3.2). The value r_{min} is the minimum value that is appropriate for n, the order of the distinguished point P in the system parameters. For this Standard, $r_{min} > 2^{160}$. For example, if the order of the underlying field is 2^{191}, an appropriate value for r_{min} is $\approx 2^{185}$. When the order of the underlying field is larger, a larger r_{min} and therefore a larger n is appropriate. Mitigating the choice

is the fact that finding a curve satisfying stricter requirements will take longer. The trial division bound l_{max} is the maximum size of all prime factors of the cofactor h. In this Standard, the order of an elliptic curve will be a number u such that $u = hn$, where n is a large prime factor (and the order of the distinguished point P) and h is a number whose prime factors are all less than l_{max}. For example, if the order of the underlying field is 2^{191} and r_{min} is 2^{185}, then an appropriate value for l_{max} is 255.

D.7. Repeated Cryptovariables

If two users are using the same system parameters and somehow generate identical private key d values, then either could impersonate the other. As the private key d is a value between 1 and n-1 (inclusive) and n is required to be greater than 2^{160}, a duplicate private key is only expected to happen by chance (due to the birthday phenomenon) after about 2^{80} key pairs have been generated. As 2^{80} is over 1 million million million million, this is not expected to happen. However, it is possible that a private key might repeat due to a hardware or software error or a poorly-seeded pseudorandom number generator. If this occurred, the public key Q for the two users would also repeat. To address the possibility of error, a service that a Certificate Authority may choose to provide for users with high security requirements is to monitor public keys to ensure that there are no duplicates. If a duplicate public key is detected, then both parties should separately be told to revoke their current public key, determine if there has been an error, try to determine the cause of the error, decide what corrective action to take (if any), and regenerate new key pairs.

The possibility of a per-message secret k value repeating during signature generation may also be a concern. A k value has the same numeric and security constraints as a private key. If a k value repeats for two different messages, then the r value in the signature will also repeat and it is then possible for an adversary with access to both signatures to recover the associated private key. As with the private key, this event should never occur except by chance. One way to address the possibility of an otherwise undetected hardware or software error or a poorly-seeded pseudorandom number generator is for a system intended for users with high security requirements to maintain a list of r values previously output by signature generation so that it can detect if an r value ever repeats. If a repeated r value is detected, the associated signature should not be output and a possible error indicated. The owner of the system should try to determine what happened and what corrective action to take, including whether to continue to operate the system.

D.8. Attacks on the Hash Function

This standard specifies the use of the Secure Hash Algorithm Revision 1 (SHA-1). If SHA-1 is broken, this Standard should not be used as it is currently written.

D.9. Security Non-Considerations

There are several choices that may be made in this Standard that do not affect security.

Other considerations, such as performance or bandwidth, may be used to determine the requirements of an implementation.

1. The choice of basis representation for the finite field F_{2^m} is not a security consideration. That is, polynomial bases and optimal normal bases are equally secure. In fact, because an appropriate matrix multiplication transforms one representation into the other (see Section G.2.3), a valid implementation choice to meet a specific output basis requirement is to use one basis internally to do the calculations and then transform the results into the other basis by using the proper change-of-basis matrix.
2. The choice of the base point P is not a security consideration as long as it has a large prime order as required by this Standard. That is, for a given elliptic curve, there are many equally-secure possibilities for the distinguished point P.
3. The representation of a point in a compressed or uncompressed form is not a security consideration.

Appendix E Tables of Trinomials, Pentanomials, and Optimal Normal Bases

[Informative]

E.1. Table of Fields F_{2^m} which have an ONB

Table E-1: Values of m, $160 \le m \le 2000$, for which the field F_{2^m} has an ONB over F_2.

162*	293	429	585	741	873	1041	1218	1359	1530*	1746*	1900*
172*	299	431	586*	743	876*	1043	1223	1370	1533	1749	1901
173	303	438	593	746	879	1049	1228*	1372*	1539	1755	1906*
174	306	441	606	749	882*	1055	1229	1380*	1541	1758	1923
178*	309	442*	611	755	891	1060*	1233	1394	1548*	1763	1925
179	316*	443	612*	756*	893	1065	1236*	1398	1559	1766	1926
180*	323	453	614	761	906*	1070	1238	1401	1570*	1769	1930*
183	326	460*	615	765	911	1090*	1251	1409	1583	1773	1931
186	329	466*	618+	771	923	1103	1258*	1418	1593	1778	1938
189	330	470	629	772*	930	1106	1265	1421	1601	1779	1948*
191	338	473	638	774	933	1108*	1269	1425	1618*	1785	1953
194	346*	483	639	779	935	1110	1271	1426*	1620*	1786*	1955
196*	348*	490*	641	783	938	1116*	1274	1430	1626	1790	1958
209	350	491	645	785	939	1118	1275	1439	1636*	1791	1959
210+	354	495	650	786*	940*	1119	1276*	1443	1649	1806	1961
221	359	508*	651	791	946*	1121	1278	1450*	1653	1811	1965
226*	371	509	652*	796*	950	1122*	1282*	1451	1659	1818	1972*
230	372*	515	653	803	953	1133	1289	1452*	1661	1821	1973
231	375	519	658*	809	965	1134	1290*	1454	1666*	1829	1978*
233	378+	522*	659	810	974	1146	1295	1463	1668*	1835	1983
239	386	530	660*	818	975	1154	1300*	1469	1673	1838	1986*
243	388*	531	676*	820*	986	1155	1306*	1478	1679	1845	1994
245	393	540*	683	826*	989	1166	1310	1481	1685	1850	1996*
251	398	543	686	828*	993	1169	1323	1482*	1692*	1854	
254	410	545	690	831	998	1170*	1329	1492*	1703	1859	
261	411	546*	700*	833	1013	1178	1331	1498*	1706	1860*	
268*	413	554	708*	834	1014	1185	1338	1499	1730	1863	
270	414	556*	713	846	1018*	1186*	1341	1505	1732*	1866+	
273	418*	558	719	852*	1019	1194	1346	1509	1733	1876*	
278	419	561	723	858*	1026	1199	1349	1511	1734	1883	
281	420*	562*	725	866	1031	1211	1353	1518	1740*	1889	
292*	426	575	726	870	1034	1212*	1355	1522*	1745	1898	

* indicates the existence of a type I ONB; + indicates the existence of both a type I and a type II ONB; otherwise there exists only a type II ONB.

E.2. Irreducible Trinomials Over F_2

Table E-2: Irreducible trinomials $x^m + x^k + 1$ over F_2.

For each m, $160 \le m \le 609$, for which an irreducible trinomial of degree m exists, the table lists the smallest k for which $x^m + x^k + 1$ is irreducible over F_2.

m	k	m	k	m	k	m	k	m	k	m	k
161	18	236	5	308	15	383	90	458	203	527	47
162	27	238	73	310	93	385	6	460	19	529	42
166	37	239	36	313	79	386	83	462	73	532	1
167	6	241	70	314	15	388	159	463	93	534	161
169	34	242	95	316	63	390	9	465	31	537	94
170	11	244	111	318	45	391	28	468	27	538	195
172	1	247	82	319	36	393	7	470	9	540	9
174	13	249	35	321	31	394	135	471	1	543	16
175	6	250	103	322	67	396	25	473	200	545	122
177	8	252	15	324	51	399	26	474	191	550	193
178	31	253	46	327	34	401	152	476	9	551	135
180	3	255	52	329	50	402	171	478	121	553	39
182	81	257	12	330	99	404	65	479	104	556	153
183	56	258	71	332	89	406	141	481	138	558	73
185	24	260	15	333	2	407	71	484	105	559	34
186	11	263	93	337	55	409	87	486	81	561	71
191	9	265	42	340	45	412	147	487	94	564	163
193	15	266	47	342	125	414	13	489	83	566	153
194	87	268	25	343	75	415	102	490	219	567	28
196	3	270	53	345	22	417	107	492	7	569	77
198	9	271	58	346	63	418	199	494	17	570	67
199	34	273	23	348	103	420	7	495	76	574	13
201	14	274	67	350	53	422	149	497	78	575	146
202	55	276	63	351	34	423	25	498	155	577	25
204	27	278	5	353	69	425	12	500	27	580	237
207	43	279	5	354	99	426	63	503	3	582	85
209	6	281	93	358	57	428	105	505	156	583	130
210	7	282	35	359	68	431	120	506	23	585	88
212	105	284	53	362	63	433	33	508	9	588	35
214	73	286	69	364	9	436	165	510	69	590	93
215	23	287	71	366	29	438	65	511	10	593	86
217	45	289	21	367	21	439	49	513	26	594	19
218	11	292	37	369	91	441	7	514	67	596	273
220	7	294	33	370	139	444	81	516	21	599	30
223	33	295	48	372	111	446	105	518	33	601	201
225	32	297	5	375	16	447	73	519	79	602	215
228	113	300	5	377	41	449	134	521	32	604	105
231	26	302	41	378	43	450	47	522	39	606	165
233	74	303	1	380	47	455	38	524	167	607	105
234	31	305	102	382	81	457	16	526	97	609	31

Table E-2.a: Irreducible trinomials $x^m + x^k + 1$ over F_2.

For each m, $610 \leq m \leq 1060$, for which an irreducible trinomial of degree m exists, the table lists the smallest k for which $x^m + x^k + 1$ is irreducible over F_2.

m	k	m	k	m	k	m	k	m	k	m	k
610	127	684	209	754	19	833	149	903	35	988	121
612	81	686	197	756	45	834	15	905	117	990	161
614	45	687	13	758	233	838	61	906	123	991	39
615	211	689	14	759	98	839	54	908	143	993	62
617	200	690	79	761	3	841	144	911	204	994	223
618	295	692	299	762	83	842	47	913	91	996	65
620	9	694	169	767	168	844	105	916	183	998	101
622	297	695	177	769	120	845	2	918	77	999	59
623	68	697	267	772	7	846	105	919	36	1001	17
625	133	698	215	774	185	847	136	921	221	1007	75
626	251	700	75	775	93	849	253	924	31	1009	55
628	223	702	37	777	29	850	111	926	365	1010	99
631	307	705	17	778	375	852	159	927	403	1012	115
633	101	708	15	780	13	855	29	930	31	1014	385
634	39	711	92	782	329	857	119	932	177	1015	186
636	217	713	41	783	68	858	207	935	417	1020	135
639	16	714	23	785	92	860	35	937	217	1022	317
641	11	716	183	791	30	861	14	938	207	1023	7
642	119	718	165	793	253	862	349	942	45	1025	294
646	249	719	150	794	143	865	1	943	24	1026	35
647	5	721	9	798	53	866	75	945	77	1028	119
649	37	722	231	799	25	868	145	948	189	1029	98
650	3	724	207	801	217	870	301	951	260	1030	93
651	14	726	5	804	75	871	378	953	168	1031	68
652	93	727	180	806	21	873	352	954	131	1033	108
654	33	729	58	807	7	876	149	956	305	1034	75
655	88	730	147	809	15	879	11	959	143	1036	411
657	38	732	343	810	159	881	78	961	18	1039	21
658	55	735	44	812	29	882	99	964	103	1041	412
660	11	737	5	814	21	884	173	966	201	1042	439
662	21	738	347	815	333	887	147	967	36	1044	41
663	107	740	135	817	52	889	127	969	31	1047	10
665	33	742	85	818	119	890	183	972	7	1049	141
668	147	743	90	820	123	892	31	975	19	1050	159
670	153	745	258	822	17	894	173	977	15	1052	291
671	15	746	351	823	9	895	12	979	178	1054	105
673	28	748	19	825	38	897	113	982	177	1055	24
676	31	750	309	826	255	898	207	983	230	1057	198
679	66	751	18	828	189	900	1	985	222	1058	27
682	171	753	158	831	49	902	21	986	3	1060	439

Table E-2.b: Irreducible trinomials $x^m + x^k + 1$ over F_2.

For each m, $1061 \leq m \leq 1516$, for which an irreducible trinomial of degree m exists, the table lists the smallest k for which $x^m + x^k + 1$ is irreducible over F_2.

m	k	m	k	m	k	m	k	m	k	m	k
1062	49	1140	141	1212	203	1287	470	1366	1	1441	322
1063	168	1142	357	1214	257	1289	99	1367	134	1442	395
1065	463	1145	277	1215	302	1294	201	1369	88	1444	595
1071	7	1146	131	1217	393	1295	38	1372	181	1446	421
1078	361	1148	23	1218	91	1297	198	1374	609	1447	195
1079	230	1151	90	1220	413	1298	399	1375	52	1449	13
1081	24	1153	241	1223	255	1300	75	1377	100	1452	315
1082	407	1154	75	1225	234	1302	77	1380	183	1454	297
1084	189	1156	307	1226	167	1305	326	1383	130	1455	52
1085	62	1158	245	1228	27	1306	39	1385	12	1457	314
1086	189	1159	66	1230	433	1308	495	1386	219	1458	243
1087	112	1161	365	1231	105	1310	333	1388	11	1460	185
1089	91	1164	19	1233	151	1311	476	1390	129	1463	575
1090	79	1166	189	1234	427	1313	164	1391	3	1465	39
1092	23	1167	133	1236	49	1314	19	1393	300	1466	311
1094	57	1169	114	1238	153	1319	129	1396	97	1468	181
1095	139	1170	27	1239	4	1321	52	1398	601	1470	49
1097	14	1174	133	1241	54	1324	337	1399	55	1471	25
1098	83	1175	476	1242	203	1326	397	1401	92	1473	77
1100	35	1177	16	1246	25	1327	277	1402	127	1476	21
1102	117	1178	375	1247	14	1329	73	1404	81	1478	69
1103	65	1180	25	1249	187	1332	95	1407	47	1479	49
1105	21	1182	77	1252	97	1334	617	1409	194	1481	32
1106	195	1183	87	1255	589	1335	392	1410	383	1482	411
1108	327	1185	134	1257	289	1337	75	1412	125	1486	85
1110	417	1186	171	1260	21	1338	315	1414	429	1487	140
1111	13	1188	75	1263	77	1340	125	1415	282	1489	252
1113	107	1190	233	1265	119	1343	348	1417	342	1490	279
1116	59	1191	196	1266	7	1345	553	1420	33	1492	307
1119	283	1193	173	1268	345	1348	553	1422	49	1495	94
1121	62	1196	281	1270	333	1350	237	1423	15	1497	49
1122	427	1198	405	1271	17	1351	39	1425	28	1500	25
1126	105	1199	114	1273	168	1353	371	1426	103	1503	80
1127	27	1201	171	1276	217	1354	255	1428	27	1505	246
1129	103	1202	287	1278	189	1356	131	1430	33	1508	599
1130	551	1204	43	1279	216	1358	117	1431	17	1510	189
1134	129	1206	513	1281	229	1359	98	1433	387	1511	278
1135	9	1207	273	1282	231	1361	56	1434	363	1513	399
1137	277	1209	118	1284	223	1362	655	1436	83	1514	299
1138	31	1210	243	1286	153	1364	239	1438	357	1516	277

Table E-2.c: Irreducible trinomials $x^m + x^k + 1$ over F_2.

For each m, $1517 \le m \le 2000$, for which an irreducible trinomial of degree m exists, the table lists the smallest k for which $x^m + x^k + 1$ is irreducible over F_2.

m	k	m	k	m	k	m	k	m	k	m	k
1518	69	1590	169	1673	90	1756	99	1838	53	1927	25
1519	220	1591	15	1674	755	1759	165	1839	836	1929	31
1521	229	1593	568	1376	363	1764	105	1841	66	1932	277
1524	27	1596	3	1678	129	1767	250	1844	339	1934	413
1526	473	1599	643	1679	20	1769	327	1846	901	1935	103
1527	373	1601	548	1681	135	1770	279	1847	180	1937	231
1529	60	1602	783	1687	31	1772	371	1849	49	1938	747
1530	207	1604	317	1689	758	1774	117	1854	885	1940	113
1534	225	1606	153	1692	359	1775	486	1855	39	1943	11
1535	404	1607	87	1694	501	1777	217	1857	688	1945	91
1537	46	1609	231	1695	29	1778	635	1860	13	1946	51
1540	75	1612	771	1697	201	1780	457	1862	149	1948	603
1542	365	1615	103	1698	459	1782	57	1863	260	1950	9
1543	445	1617	182	1700	225	1783	439	1865	53	1951	121
1545	44	1618	211	1703	161	1785	214	1866	11	1953	17
1548	63	1620	27	1705	52	1788	819	1870	121	1956	279
1550	189	1623	17	1708	93	1790	593	1871	261	1958	89
1551	557	1625	69	1710	201	1791	190	1873	199	1959	371
1553	252	1628	603	1711	178	1793	114	1878	253	1961	771
1554	99	1630	741	1713	250	1798	69	1879	174	1962	99
1556	65	1631	668	1716	221	1799	312	1881	370	1964	21
1558	9	1633	147	1719	113	1801	502	1884	669	1966	801
1559	119	1634	227	1721	300	1802	843	1886	833	1967	26
1561	339	1636	37	1722	39	1804	747	1887	353	1969	175
1562	95	1638	173	1724	261	1806	101	1889	29	1974	165
1564	7	1639	427	1726	753	1807	123	1890	371	1975	841
1566	77	1641	287	1729	94	1809	521	1895	873	1977	238
1567	127	1642	231	1734	461	1810	171	1900	235	1980	33
1569	319	1647	310	1735	418	1814	545	1902	733	1983	113
1570	667	1649	434	1737	403	1815	163	1903	778	1985	311
1572	501	1650	579	1738	267	1817	479	1905	344	1986	891
1575	17	1652	45	1740	259	1818	495	1906	931	1988	555
1577	341	1655	53	1742	869	1820	11	1908	945	1990	133
1578	731	1657	16	1743	173	1823	684	1911	67	1991	546
1580	647	1660	37	1745	369	1825	9	1913	462	1993	103
1582	121	1663	99	1746	255	1828	273	1918	477	1994	15
1583	20	1665	176	1748	567	1830	381	1919	105	1996	307
1585	574	1666	271	1750	457	1831	51	1921	468	1999	367
1586	399	1668	459	1751	482	1833	518	1924	327		
1588	85	1671	202	1753	775	1836	243	1926	357		

E.3. Irreducible Pentanomials Over F_2

Table E-3: Irreducible pentanomials $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$ over F_2.
For each m, $160 \le m \le 488$, for which an irreducible trinomial of degree m does not exist, a triple of exponents k_1, k_2, k_3 is given for which the pentanomial $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, is irreducible over F_2.

m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)
160	1, 2, 117	243	1, 2, 17	326	1, 2, 67	410	1, 2, 16
163	1, 2, 8	245	1, 2, 37	328	1, 2, 51	411	1, 2, 50
164	1, 2, 49	246	1, 2, 11	331	1, 2, 134	413	1, 2, 33
165	1, 2, 25	248	1, 2, 243	334	1, 2, 5	416	1, 3, 76
168	1, 2, 65	251	1, 2, 45	335	1, 2, 250	419	1, 2, 129
171	1, 3, 42	254	1, 2, 7	336	1, 2, 77	421	1, 2, 81
173	1, 2, 10	256	1, 2, 155	338	1, 2, 112	424	1, 2, 177
176	1, 2, 43	259	1, 2, 254	339	1, 2, 26	427	1, 2, 245
179	1, 2, 4	261	1, 2, 74	341	1, 2, 57	429	1, 2, 14
181	1, 2, 89	262	1, 2, 207	344	1, 2, 7	430	1, 2, 263
184	1, 2, 81	264	1, 2, 169	347	1, 2, 96	432	1, 2, 103
187	1, 2, 20	267	1, 2, 29	349	1, 2, 186	434	1, 2, 64
188	1, 2, 60	269	1, 2, 117	352	1, 2, 263	435	1, 2, 166
189	1, 2, 49	272	1, 3, 56	355	1, 2, 138	437	1, 2, 6
190	1, 2, 47	275	1, 2, 28	356	1, 2, 69	440	1, 2, 37
192	1, 2, 7	277	1, 2, 33	357	1, 2, 28	442	1, 2, 32
195	1, 2, 37	280	1, 2, 113	360	1, 2, 49	443	1, 2, 57
197	1, 2, 21	283	1, 2, 200	361	1, 2, 44	445	1, 2, 225
200	1, 2, 81	285	1, 2, 77	363	1, 2, 38	448	1, 3, 83
203	1, 2, 45	288	1, 2, 191	365	1, 2, 109	451	1, 2, 33
205	1, 2, 21	290	1, 2, 70	368	1, 2, 85	452	1, 2, 10
206	1, 2, 63	291	1, 2, 76	371	1, 2, 156	453	1, 2, 88
208	1, 2, 83	293	1, 3, 154	373	1, 3, 172	454	1, 2, 195
211	1, 2, 165	296	1, 2, 123	374	1, 2, 109	456	1, 2, 275
213	1, 2, 62	298	1, 2, 78	376	1, 2, 77	459	1, 2, 332
216	1, 2, 107	299	1, 2, 21	379	1, 2, 222	461	1, 2, 247
219	1, 2, 65	301	1, 2, 26	381	1, 2, 5	464	1, 2, 310
221	1, 2, 18	304	1, 2, 11	384	1, 2, 299	466	1, 2, 78
222	1, 2, 73	306	1, 2, 106	387	1, 2, 146	467	1, 2, 210
224	1, 2, 159	307	1, 2, 93	389	1, 2, 159	469	1, 2, 149
226	1, 2, 30	309	1, 2, 26	392	1, 2, 145	472	1, 2, 33
227	1, 2, 21	311	1, 3, 155	395	1, 2, 333	475	1, 2, 68
229	1, 2, 21	312	1, 2, 83	397	1, 2, 125	477	1, 2, 121
230	1, 2, 13	315	1, 2, 142	398	1, 3, 23	480	1, 2, 149
232	1, 2, 23	317	1, 3, 68	400	1, 2, 245	482	1, 2, 13
235	1, 2, 45	320	1, 2, 7	403	1, 2, 80	483	1, 2, 352
237	1, 2, 104	323	1, 2, 21	405	1, 2, 38	485	1, 2, 70
240	1, 3, 49	325	1, 2, 53	408	1, 2, 323	488	1, 2, 123

Table E-3.a: Irreducible pentanomials $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$ over F_2.
For each m, $490 \le m \le 811$, for which an irreducible trinomial of degree m does not exist, a triple of exponents k_1, k_2, k_3 is given for which the pentanomial $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, is irreducible over F_2.

m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)
491	1, 2, 270	571	1, 2, 408	653	1, 2, 37	734	1, 2, 67
493	1, 2, 171	572	1, 2, 238	656	1, 2, 39	736	1, 2, 359
496	1, 3, 52	573	1, 2, 220	659	1, 2, 25	739	1, 2, 60
499	1, 2, 174	576	1, 3, 52	661	1, 2, 80	741	1, 2, 34
501	1, 2, 332	578	1, 2, 138	664	1, 2, 177	744	1, 2, 347
502	1, 2, 99	579	1, 3, 526	666	1, 2, 100	747	1, 2, 158
504	1, 3, 148	581	1, 2, 138	667	1, 2, 161	749	1, 2, 357
507	1, 2, 26	584	1, 2, 361	669	1, 2, 314	752	1, 2, 129
509	1, 2, 94	586	1, 2, 14	672	1, 2, 91	755	1, 4, 159
512	1, 2, 51	587	1, 2, 130	674	1, 2, 22	757	1, 2, 359
515	1, 2, 73	589	1, 2, 365	675	1, 2, 214	760	1, 2, 17
517	1, 2, 333	591	1, 2, 38	677	1, 2, 325	763	1, 2, 17
520	1, 2, 291	592	1, 2, 143	678	1, 2, 95	764	1, 2, 12
523	1, 2, 66	595	1, 2, 9	680	1, 2, 91	765	1, 2, 137
525	1, 2, 92	597	1, 2, 64	681	1, 2, 83	766	1, 3, 280
528	1, 2, 35	598	1, 2, 131	683	1, 2, 153	768	1, 2, 115
530	1, 2, 25	600	1, 2, 239	685	1, 3, 4	770	1, 2, 453
531	1, 2, 53	603	1, 2, 446	688	1, 2, 71	771	1, 2, 86
533	1, 2, 37	605	1, 2, 312	691	1, 2, 242	773	1, 2, 73
535	1, 2, 143	608	1, 2, 213	693	1, 2, 250	776	1, 2, 51
536	1, 2, 165	611	1, 2, 13	696	1, 2, 241	779	1, 2, 456
539	1, 2, 37	613	1, 2, 377	699	1, 2, 40	781	1, 2, 209
541	1, 2, 36	616	1, 2, 465	701	1, 2, 466	784	1, 2, 59
542	1, 3, 212	619	1, 2, 494	703	1, 2, 123	786	1, 2, 118
544	1, 2, 87	621	1, 2, 17	704	1, 2, 277	787	1, 2, 189
546	1, 2, 8	624	1, 2, 71	706	1, 2, 27	788	1, 2, 375
547	1, 2, 165	627	1, 2, 37	707	1, 2, 141	789	1, 2, 5
548	1, 2, 385	629	1, 2, 121	709	1, 2, 9	790	1, 2, 111
549	1, 3, 274	630	1, 2, 49	710	1, 3, 29	792	1, 2, 403
552	1, 2, 41	632	1, 2, 9	712	1, 2, 623	795	1, 2, 137
554	1, 2, 162	635	1, 2, 64	715	1, 3, 458	796	1, 2, 36
555	1, 2, 326	637	1, 2, 84	717	1, 2, 320	797	1, 2, 193
557	1, 2, 288	638	1, 2, 127	720	1, 2, 625	800	1, 2, 463
560	1, 2, 157	640	1, 3, 253	723	1, 2, 268	802	1, 2, 102
562	1, 2, 56	643	1, 2, 153	725	1, 2, 331	803	1, 2, 208
563	1, 4, 159	644	1, 2, 24	728	1, 2, 51	805	1, 2, 453
565	1, 2, 66	645	1, 2, 473	731	1, 2, 69	808	1, 3, 175
568	1, 2, 291	648	1, 2, 235	733	1, 2, 92	811	1, 2, 18

Table E-3.b: Irreducible pentanomials $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$ over F_2.
For each m, $812 \le m \le 1131$, for which an irreducible trinomial of degree m does not exist, a triple of exponents k_1, k_2, k_3 is given for which the pentanomial $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, is irreducible over F_2.

m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)
813	1, 2, 802	901	1, 2, 581	973	1, 2, 113	1053	1, 2, 290
816	1, 3, 51	904	1, 3, 60	974	1, 2, 211	1056	1, 2, 11
819	1, 2, 149	907	1, 3, 26	976	1, 2, 285	1059	1, 3, 6
821	1, 2, 177	909	1, 3, 168	978	1, 2, 376	1061	1, 2, 166
824	1, 2, 495	910	1, 2, 357	980	1, 2, 316	1064	1, 2, 946
827	1, 2, 189	912	1, 2, 569	981	1, 2, 383	1066	1, 2, 258
829	1, 2, 560	914	1, 2, 4	984	1, 2, 349	1067	1, 2, 69
830	1, 2, 241	915	1, 2, 89	987	1, 3, 142	1068	1, 2, 223
832	1, 2, 39	917	1, 2, 22	989	1, 2, 105	1069	1, 2, 146
835	1, 2, 350	920	1, 3, 517	992	1, 2, 585	1070	1, 3, 94
836	1, 2, 606	922	1, 2, 24	995	1, 3, 242	1072	1, 2, 443
837	1, 2, 365	923	1, 2, 142	997	1, 2, 453	1073	1, 3, 235
840	1, 2, 341	925	1, 2, 308	1000	1, 3, 68	1074	1, 2, 395
843	1, 2, 322	928	1, 2, 33	1002	1, 2, 266	1075	1, 2, 92
848	1, 2, 225	929	1, 2, 36	1003	1, 2, 410	1076	1, 2, 22
851	1, 2, 442	931	1, 2, 72	1004	1, 2, 96	1077	1, 2, 521
853	1, 2, 461	933	1, 2, 527	1005	1, 2, 41	1080	1, 2, 151
854	1, 2, 79	934	1, 3, 800	1006	1, 2, 63	1083	1, 2, 538
856	1, 2, 842	936	1, 3, 27	1008	1, 2, 703	1088	1, 2, 531
859	1, 2, 594	939	1, 2, 142	1011	1, 2, 17	1091	1, 2, 82
863	1, 2, 90	940	1, 2, 204	1013	1, 2, 180	1093	1, 2, 173
864	1, 2, 607	941	1, 2, 573	1016	1, 2, 49	1096	1, 2, 351
867	1, 2, 380	944	1, 2, 487	1017	1, 2, 746	1099	1, 2, 464
869	1, 2, 82	946	1, 3, 83	1018	1, 2, 27	1101	1, 2, 14
872	1, 2, 691	947	1, 2, 400	1019	1, 2, 96	1104	1, 2, 259
874	1, 2, 110	949	1, 2, 417	1021	1, 2, 5	1107	1, 2, 176
875	1, 2, 66	950	1, 2, 859	1024	1, 2, 515	1109	1, 2, 501
877	1, 2, 140	952	1, 3, 311	1027	1, 2, 378	1112	1, 2, 1045
878	1, 2, 343	955	1, 2, 606	1032	1, 2, 901	1114	1, 2, 345
880	1, 3, 221	957	1, 2, 158	1035	1, 2, 76	1115	1, 2, 268
883	1, 2, 488	958	1, 2, 191	1037	1, 2, 981	1117	1, 2, 149
885	1, 2, 707	960	1, 2, 491	1038	1, 2, 41	1118	1, 2, 475
886	1, 2, 227	962	1, 2, 18	1040	1, 2, 429	1120	1, 3, 386
888	1, 2, 97	963	1, 2, 145	1043	1, 3, 869	1123	1, 2, 641
891	1, 2, 364	965	1, 2, 213	1045	1, 2, 378	1124	1, 2, 156
893	1, 2, 13	968	1, 2, 21	1046	1, 2, 39	1125	1, 2, 206
896	1, 2, 19	970	1, 2, 260	1048	1, 3, 172	1128	1, 3, 7
899	1, 3, 898	971	1, 2, 6	1051	1, 3, 354	1131	1, 2, 188

Table E-3.c: Irreducible pentanomials $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$ over F_2.
For each m, $1131 \le m \le 1456$, for which an irreducible trinomial of degree m does not exist, a triple of exponents k_1, k_2, k_3 is given for which the pentanomial $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, is irreducible over F_2.

m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)	M	k or (k_1, k_2, k_3)
1132	1, 2, 20	1219	1, 2, 225	1296	1, 2, 379	1376	1, 2, 1201
1133	1, 2, 667	1221	1, 2, 101	1299	1, 2, 172	1378	1, 2, 362
1136	1, 2, 177	1222	1, 2, 215	1301	1, 2, 297	1379	1, 2, 400
1139	1, 2, 45	1224	1, 2, 157	1303	1, 2, 306	1381	1, 2, 56
1141	1 2 134	1227	1, 2, 361	1304	1, 3, 574	1382	1, 3, 58
1143	1, 2, 7	1229	1, 2, 627	1307	1, 2, 157	1384	1, 2, 1131
1144	1, 2, 431	1232	1, 2, 225	1309	1, 2, 789	1387	1, 2, 33
1147	1, 2, 390	1235	1, 2, 642	1312	1, 2, 1265	1389	1, 2, 41
1149	1, 2, 221	1237	1, 2, 150	1315	1, 2, 270	1392	1, 2, 485
1150	1, 2, 63	1240	1, 2, 567	1316	1, 2, 12	1394	1, 2, 30
1152	1, 2, 971	1243	1, 2, 758	1317	1, 2, 254	1395	1, 2, 233
1155	1, 2, 94	1244	1, 2, 126	1318	1, 3, 94	1397	1, 2, 397
1157	1, 2, 105	1245	1, 2, 212	1320	1, 2, 835	1400	1, 2, 493
1160	1, 2, 889	1248	1, 2, 1201	1322	1, 2, 538	1403	1, 2, 717
1162	1, 2, 288	1250	1, 2, 37	1323	1, 2, 1198	1405	1, 2, 558
1163	1, 2, 33	1251	1, 2, 1004	1325	1, 2, 526	1406	1, 2, 13
1165	1, 2, 494	1253	1, 2, 141	1328	1, 2, 507	1408	1, 3, 45
1168	1, 2, 473	1254	1, 2, 697	1330	1, 2, 609	1411	1, 2, 200
1171	1, 2, 396	1256	1, 2, 171	1331	1, 2, 289	1413	1, 2, 101
1172	1, 2, 426	1258	1, 2, 503	1333	1, 2, 276	1416	1, 3, 231
1173	1, 2, 673	1259	1, 2, 192	1336	1, 2, 815	1418	1, 2, 283
1176	1, 2, 19	1261	1, 2, 14	1339	1, 2, 284	1419	1, 2, 592
1179	1, 2, 640	1262	1, 2, 793	1341	1, 2, 53	1421	1, 2, 30
1181	1, 2, 82	1264	1, 2, 285	1342	1, 2, 477	1424	1, 2, 507
1184	1, 2, 1177	1267	1, 2, 197	1344	1, 2, 469	1427	1, 2, 900
1187	1, 2, 438	1269	1, 2, 484	1346	1, 2, 57	1429	1, 2, 149
1189	1, 2, 102	1272	1, 2, 223	1347	1, 2, 61	1432	1, 2, 251
1192	1, 3, 831	1274	1, 2, 486	1349	1, 2, 40	1435	1, 2, 126
1194	1, 2, 317	1275	1, 2, 25	1352	1, 2, 583	1437	1, 2, 545
1195	1, 2, 293	1277	1, 2, 451	1355	1, 2, 117	1439	1, 2, 535
1197	1, 2, 269	1280	1, 2, 843	1357	1, 2, 495	1440	1, 3, 1023
1200	1, 3, 739	1283	1, 2, 70	1360	1, 2, 393	1443	1, 2, 413
1203	1, 2, 226	1285	1, 2, 564	1363	1, 2, 852	1445	1, 2, 214
1205	1, 2, 4	1288	1, 2, 215	1365	1, 2, 329	1448	1, 3, 212
1208	1, 2, 915	1290	1, 2, 422	1368	1, 2, 41	1450	1, 2, 155
1211	1, 2, 373	1291	1, 2, 245	1370	1, 2, 108	1451	1, 2, 193
1213	1, 2, 245	1292	1, 2, 78	1371	1, 2, 145	1453	1, 2, 348
1216	1, 2, 155	1293	1, 2, 26	1373	1, 2, 613	1456	1, 2, 1011

Table E-3.d: Irreducible pentanomials $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$ over F_2.
For each m, $1458 \le m \le 1761$, for which an irreducible trinomial of degree m does not exist, a triple of exponents k_1, k_2, k_3 is given for which the pentanomial $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, is irreducible over F_2.

m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)	M	k or (k_1, k_2, k_3)
1459	1, 2, 1032	1536	1, 2, 881	1619	1, 2, 289	1690	1, 2, 200
1461	1, 2, 446	1538	1, 2, 6	1621	1, 2, 1577	1691	1, 2, 556
1462	1, 2, 165	1539	1, 2, 80	1622	1, 2, 1341	1693	1, 2, 137
1464	1, 2, 275	1541	1, 2, 4	1624	1, 2, 1095	1696	1, 2, 737
1467	1, 2, 113	1544	1, 2, 99	1626	1, 2, 191	1699	1, 2, 405
1469	1, 2, 775	1546	1, 2, 810	1627	1, 2, 189	1701	1, 2, 568
1472	1, 2, 613	1547	1, 2, 493	1629	1, 2, 397	1702	1, 2, 245
1474	1, 2, 59	1549	1, 2, 426	1632	1, 2, 211	1704	1, 3, 55
1475	1, 2, 208	1552	1, 2, 83	1635	1, 2, 113	1706	1, 2, 574
1477	1, 2, 1325	1555	1, 2, 254	1637	1, 2, 234	1707	1, 2, 221
1480	1, 2, 285	1557	1, 2, 20	1640	1, 2, 715	1709	1, 2, 201
1483	1, 2, 1077	1560	1, 2, 11	1643	1, 2, 760	1712	1, 2, 445
1484	1, 2, 61	1563	1, 2, 41	1644	1, 2, 236	1714	1, 2, 191
1485	1, 2, 655	1565	1, 2, 18	1645	1, 2, 938	1715	1, 2, 612
1488	1, 2, 463	1568	1, 2, 133	1646	1, 2, 435	1717	1, 2, 881
1491	1, 2, 544	1571	1, 2, 21	1648	1, 2, 77	1718	1, 2, 535
1493	1, 2, 378	1573	1, 2, 461	1651	1, 2, 873	1720	1, 2, 525
1494	1, 2, 731	1574	1, 2, 331	1653	1, 2, 82	1723	1, 2, 137
1496	1, 2, 181	1576	1, 2, 147	1654	1, 3, 201	1725	1, 2, 623
1498	1, 2, 416	1579	1, 2, 374	1656	1, 2, 361	1727	1, 2, 22
1499	1, 2, 477	1581	1, 2, 160	1658	1, 2, 552	1728	1, 2, 545
1501	1, 2, 60	1584	1, 2, 895	1659	1, 2, 374	1730	1, 2, 316
1502	1, 2, 111	1587	1, 2, 433	1661	1, 2, 84	1731	1, 2, 925
1504	1, 2, 207	1589	1, 2, 882	1662	1, 3, 958	1732	1, 2, 75
1506	1, 2, 533	1592	1, 2, 223	1664	1, 2, 399	1733	1, 2, 285
1507	1, 2, 900	1594	1, 2, 971	1667	1, 2, 1020	1736	1, 2, 435
1509	1, 2, 209	1595	1, 2, 18	1669	1, 2, 425	1739	1, 2, 409
1512	1, 2, 1121	1597	1, 2, 42	1670	1, 2, 19	1741	1, 3, 226
1515	1, 2, 712	1598	1, 2, 385	1672	1, 2, 405	1744	1, 2, 35
1517	1, 2, 568	1600	1, 2, 57	1675	1, 2, 77	1747	1, 2, 93
1520	1, 2, 81	1603	1, 2, 917	1677	1, 2, 844	1749	1, 2, 236
1522	1, 2, 47	1605	1, 2, 46	1680	1, 2, 1549	1752	1, 2, 559
1523	1, 2, 240	1608	1, 2, 271	1682	1, 2, 354	1754	1, 2, 75
1525	1, 2, 102	1610	1, 2, 250	1683	1, 2, 1348	1755	1, 2, 316
1528	1, 2, 923	1611	1, 2, 58	1684	1, 2, 474	1757	1, 2, 21
1531	1, 2, 1125	1613	1, 2, 48	1685	1, 2, 493	1758	1, 2, 221
1532	1, 2, 466	1614	1, 2, 1489	1686	1, 2, 887	1760	1, 3, 1612
1533	1, 2, 763	1616	1, 2, 139	1688	1, 2, 921	1761	1, 2, 131

Table E-3.e: Irreducible pentanomials $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$ over F_2.

For each m, $1761 \le m \le 2000$, for which an irreducible trinomial of degree m does not exist, a triple of exponents k_1, k_2, k_3 is given for which the pentanomial $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, is irreducible over F_2.

m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)	m	k or (k_1, k_2, k_3)
1762	1, 2, 318	1826	1, 2, 298	1883	1, 2, 1062	1941	1, 2, 1133
1763	1, 2, 345	1827	1, 2, 154	1885	1, 2, 813	1942	1, 2, 147
1765	1, 2, 165	1829	1, 2, 162	1888	1, 2, 923	1944	1, 2, 617
1766	1, 2, 1029	1832	1, 3, 1078	1891	1, 2, 1766	1947	1, 2, 1162
1768	1, 2, 1403	1834	1, 2, 210	1892	1, 3, 497	1949	1, 2, 621
1771	1, 2, 297	1835	1, 2, 288	1893	1, 2, 461	1952	1, 3, 65
1773	1, 2, 50	1837	1, 2, 200	1894	1, 3, 215	1954	1, 2, 1226
1776	1, 2, 17	1840	1, 2 195	1896	1, 2, 451	1955	1, 2, 109
1779	1, 3, 1068	1842	1, 2, 799	1897	1, 2, 324	1957	1, 2, 17
1781	1, 2, 18	1843	1, 2, 872	1898	1, 2, 613	1960	1, 2, 939
1784	1, 2, 1489	1845	1, 2, 526	1899	1, 2, 485	1963	1, 2, 1137
1786	1, 2, 614	1848	1, 2, 871	1901	1, 2, 330	1965	1, 2, 364
1787	1, 2, 457	1850	1, 2, 79	1904	1, 2, 337	1968	1, 3, 922
1789	1, 2, 80	1851	1, 2, 250	1907	1, 2, 45	1970	1, 2, 388
1792	1, 2, 341	1852	1, 2, 339	1909	1, 2, 225	1971	1, 2, 100
1794	1, 2, 95	1853	1, 2, 705	1910	1, 3, 365	1972	1, 2, 474
1795	1, 2, 89	1856	1, 2, 585	1912	1, 2, 599	1973	1, 2, 438
1796	1, 2, 829	1858	1, 2, 1368	1914	1, 2, 544	1976	1, 3, 1160
1797	1, 2, 80	1859	1, 2, 120	1915	1, 2, 473	1978	1, 2, 158
1800	1, 2, 1013	1861	1, 2, 509	1916	1, 2, 502	1979	1, 2, 369
1803	1, 2, 248	1864	1, 2, 1379	1917	1, 2, 485	1981	1, 2, 96
1805	1, 2, 82	1867	1, 2, 117	1920	1, 2, 67	1982	1, 2, 1027
1808	1, 2, 25	1868	1, 2, 250	1922	1, 2, 36	1984	1, 2, 129
1811	1, 2, 117	1869	1, 2, 617	1923	1, 4, 40	1987	1, 2, 80
1812	1, 2, 758	1872	1, 3, 60	1925	1, 2, 576	1989	1, 2, 719
1813	1, 3, 884	1874	1, 2, 70	1928	1, 2, 763	1992	1, 2, 1241
1816	1, 2, 887	1875	1, 2, 412	1930	1, 2, 155	1995	1, 2, 37
1819	1, 2, 116	1876	1, 2, 122	1931	1, 2, 648	1997	1, 2, 835
1821	1, 2, 326	1877	1, 2, 796	1933	1, 2, 971	1998	1, 3, 1290
1822	1, 3, 31	1880	1, 2, 1647	1936	1, 2, 117	2000	1, 2, 981
1824	1, 2, 821	1882	1, 2, 128	1939	1, 2, 5		

E.4. Table of Fields F_{2^m} which Have Both an ONB and a TPB over F_2

Table E-4: Values of m, $160 \le m \le 2000$, for which the field F_{2^m} has both an ONB and a TPB over F_2.

162	292	431	606	743	858	1034	1170	1306	1492	1703	1926
172	303	438	612	746	866	1041	1178	1310	1505	1734	1938
174	316	441	614	756	870	1049	1185	1329	1511	1740	1948
178	329	460	615	761	873	1055	1186	1338	1518	1745	1953
180	330	470	618	772	876	1060	1199	1353	1530	1746	1958
183	346	473	639	774	879	1065	1212	1359	1548	1769	1959
186	348	490	641	783	882	1090	1218	1372	1559	1778	1961
191	350	495	650	785	906	1103	1223	1380	1570	1785	1983
194	354	508	651	791	911	1106	1228	1398	1583	1790	1986
196	359	519	652	809	930	1108	1233	1401	1593	1791	1994
209	372	522	658	810	935	1110	1236	1409	1601	1806	1996
210	375	540	660	818	938	1116	1238	1425	1618	1818	
231	378	543	676	820	953	1119	1265	1426	1620	1838	
233	386	545	686	826	975	1121	1271	1430	1636	1854	
239	388	556	690	828	986	1122	1276	1452	1649	1860	
268	393	558	700	831	993	1134	1278	1454	1666	1863	
270	414	561	708	833	998	1146	1282	1463	1668	1866	
273	418	575	713	834	1014	1154	1289	1478	1673	1889	
278	420	585	719	846	1026	1166	1295	1481	1679	1900	
281	426	593	726	852	1031	1169	1300	1482	1692	1906	

Appendix F Normative Number-Theoretic Algorithms
[Normative]

F.1. Avoiding Cryptographically Weak Curves

Two conditions, the *MOV condition* and the *Anomalous condition*, are described to ensure that a particular elliptic curve is not vulnerable to two known attacks on special instances of the elliptic curve discrete logarithm problem.

F.1.1. The MOV Condition

The reduction attack of Menezes, Okamoto and Vanstone [29] reduces the discrete logarithm problem in an elliptic curve over F_q to the discrete logarithm in the finite field F_{q^B} for some $B \geq 1$. The attack is only practical if B is small; this is not the case for most elliptic curves. The *MOV condition* ensures that an elliptic curve is not vulnerable to this reduction attack.

Before performing the algorithm, it is necessary to select an MOV threshold. This is a positive integer B such that taking discrete logarithms over F_{q^B} is at least as difficult as taking elliptic discrete logarithms over F_q. For this Standard, a value $B \geq 20$ is required. Selecting $B \geq 20$ also limits the selection of curves to non-supersingular curves (see Section D.1). This algorithm is used in elliptic curve parameter validation (see Section 5.1) and elliptic curve parameter generation (see Section F.3.2).

Input: An MOV threshold B, a prime-power q, and a prime n. (n is a prime divisor of $\#E(F_q)$, where E is an elliptic curve defined over F_q.)

Output: The message "True" if the MOV condition is satisfied for an elliptic curve over F_q with a base point of order n; the message "False" otherwise.

1. Set $t = 1$.
2. For i from 1 to B do
   - 2.1. Set $t = t \bullet q \bmod n$.
   - 2.2. If $t = 1$, then output "False" and stop.
3. Output "True".

F.1.2. The Anomalous Condition

Smart [39] and Satoh and Araki [38] showed that the elliptic curve discrete logarithm problem in anomalous curves can be efficiently solved. An elliptic curve E defined over F_q is said to be F_q-*anomalous* if $\#E(F_q) = q$. The *Anomalous condition* checks that $\#E(F_q) \neq q$; this ensures that an elliptic curve is not vulnerable to the Anomalous attack.

Input: An elliptic curve E defined over F_q, and the order $u = \#E(F_q)$.

Output: The message "True" if the Anomalous condition is satisfied for E over F_q; the message "False" otherwise.

1. If $u = q$ then output “False”; otherwise output “True”.

F.2. Primality

F.2.1. A Probabilistic Primality Test

If n is a large positive integer, the following probabilistic algorithm (the *Miller-Rabin test*) [21, p.379] will determine whether n is prime or composite, with an arbitrarily small probability of error. This algorithm is used in elliptic curve parameter validation (see Section 5.1), in elliptic curve parameter generation (see Section F.3.2), and in checking for near primality (see Section F.2.2).

Input: A large odd integer n, and a positive integer T.

Output: The message "probable prime" or "composite".

1. Compute v and an odd value for w such that $n - 1 = 2^v w$.
2. For j from 1 to T do
   - 2.1. Choose random a in the interval $[2, n - 1]$.
   - 2.2. Set $b = a^w \bmod n$.
   - 2.3. If $b = 1$ or $n - 1$, go to Step 2.6.

2.4. For i from 1 to v - 1 do

2.4.1 Set $b = b^2 \bmod n$.

2.4.2 If $b = n - 1$, go to Step 2.6.

2.4.3 If $b = 1$, output "composite" and stop.

2.4.4 Next i.

2.5. Output "composite" and stop.

2.6. Next j.

3. Output "probable prime".

If the algorithm outputs "composite", then n is a composite integer. The probability that the algorithm outputs "probable prime" when n is a composite integer is less than 2^{-2T}. Thus, the probability of an error can be made negligible by taking a large enough value for T. For this Standard, a value of $T \geq 50$ shall be used. If a deterministic test is needed, see Section F.2.3.

F.2.2. Checking for Near Primality

Given a trial division bound l_{max}, a positive integer h is said to be l_{max}-*smooth* if every prime divisor of h is at most l_{max}. Given a positive integer r_{min}, the positive integer u is said to be *nearly prime* if $u = hn$ for some probable prime value of n such that $n \geq r_{min}$ and some l_{max}-smooth integer h. The following algorithm checks for near primality. The algorithm is used in elliptic curve parameter generation (see Section F.3.2).

Input: Positive integers u, l_{max}, and r_{min}.

Output: If u is nearly prime, a probable prime $n \geq r_{min}$ and a l_{max}-smooth integer h such that $u = hn$. If u is not nearly prime, the message "not nearly prime".

1. Set $n = u$, $h = 1$.
2. For l from 2 to l_{max} do

2.1. If l is composite, then go to Step 2.3.

2.2. While (l divides n)

2.2.1 Set $n = n / l$ and $h = h \bullet l$.

2.2.2 If $n < r_{min}$, then output "not nearly prime" and stop.

2.3. Next l.

3. If n is probably prime (see Section F.2.1), then output h and n and stop.
4. Output "not nearly prime".

F.2.3. A Deterministic Primality Test

An application of elliptic curves is the devising of algorithms for *proving* the primality of an integer p which is deemed "prime" by the probabilistic test of Section F.2.1. If a deterministic primality test is desired, the test referenced in this section shall be used. The *Goldwasser-Kilian-Atkin (GKA) algorithm* produces a *primality certificate*: a collection

$C = \{C_1,...,C_s\}$

in which each component C_i consists of positive integers:

$C_i = (p_i, r_i, a_i, b_i, x_i, y_i)$,

where:

- For all i, (x_i, y_i) is a point of order r_i on the elliptic curve:

  $y^2 = x^3 + a_i x + b_i \pmod{p_i}$,
- $\sqrt{r_i} > \sqrt[4]{p_i} + 1$ for all i,
- $p_1 = n$,
- $p_{i+1} = r_i$ for $1 \leq i < s$,
- $r_s < l_{max}^2$,
- r_s is proved prime by trial division.

If a primality certificate exists for n, then n is prime by the following theorem.
THEOREM (Goldwasser-Kilian):

Let p and r be positive integers greater than 3 with $\sqrt{r} > \sqrt[4]{p} + 1$. Let a, b, x, and y be integers (mod p) such that (x, y) is a point of order r on the elliptic curve:
$y^2 = x^3 + ax + b \pmod{p}$.
Then p is prime if r is prime.
The GKA algorithm is specified in IEEE P1363 [12].

F.3. Elliptic Curve Algorithms

F.3.1. Finding a Point of Large Prime Order

If the order $\#E(F_q) = u$ of an elliptic curve E is nearly prime, the following algorithm efficiently produces a random point on E whose order is the large prime factor n of $u = hn$. The algorithm is used in elliptic curve parameter generation (see Section F.3.2).

Input: A prime n, a positive integer h not divisible by n, and an elliptic curve E over the field F_q with $\#E(F_q) = u$.

Output: If $u = hn$, a point P on E of order n. If not, the message "wrong order".

1. Generate a random point G (not $\mathcal{O}$) on E. (See Section G.3.1.)
2. Set $P = hG$.
3. If $P = \mathcal{O}$, then go to Step 1.
4. Set $Q = nP$.
5. If $Q \neq \mathcal{O}$, then output "wrong order" and stop.
6. Output P.

F.3.2. Selecting an Appropriate Curve and Point

Given a field size q, a lower bound r_{min} for the point order, and a trial division bound l_{max}, the following procedure shall be used for choosing a curve and arbitrary point. The algorithm is used to generate elliptic curve parameters (see Sections 5.1.1.a and 5.1.2.a).

Input: A field size q, lower bound r_{min}, and trial division bound l_{max}. (See the notes below for guidance on selecting r_{min} and l_{max}.)

Output: Field elements $a, b \in F_q$ which define an elliptic curve over F_q, a point P of prime order $n \geq r_{min}$ on the curve, and the cofactor $h = \#E(F_q)/n$.

1. If it is desired that an elliptic curve be generated verifiably at random, then select parameters (SEED, a, b) using the technique specified in Section F.3.3.a in the case that $q = 2^m$, or the technique specified in Section F.3.3.b in the case that $q = p$ is an odd prime. Compute the order u of the curve defined by a and b (see Note 5 below).

   Otherwise, use any alternative technique to select $a, b \in F_q$ which define an elliptic curve of known order u. (See Note 6 for one such technique.)

2. - In the case that q is a prime, verify that $(4a^3 + 27b^2) \not\equiv 0 \pmod{p}$.

     The curve equation for E is:

     $y^2 = x^3 + ax + b$.
   - In the case that $q = 2^m$, verify that $b \neq 0$. The curve equation for E is:

     $y^2 + xy = x^3 + ax^2 + b$.
3. Test u for near primality using the technique defined in Section F.2.2. If the result is "not nearly prime", then go to Step 1. Otherwise, $u = hn$ where h is l_{max}-smooth and $n \geq r_{min}$ is probably prime.
4. Check the MOV condition (see Section F.1.1) with inputs $B \geq 20$, q, and n. If the result is “False”, then go to Step 1.

   Check the Anomalous condition (see Section F.1.2). If the result is “False”, then go to Step 1.
5. Find a point P on E of order n. (See Section F.3.1.)
6. Output the curve E, the point P, the order n, and the cofactor h.

Notes:
1. r_{min} shall be selected so that $r_{min} > 2^{160}$. The security level can be increased by selecting a larger r_{min} (e.g. $r_{min} > 2^{200}$).
2. If q is prime, then the order u of an elliptic curve E over F_q satisfies $q + 1 - 2\sqrt{q} \le u \le q + 1 + 2\sqrt{q}$. Hence for a given q, r_{min} should be $\le q+1-2\sqrt{q}$.
3. If $q = 2^m$, then the order u of an elliptic curve E over F_q satisfies $q + 1 - 2\sqrt{q} \le u \le q + 1 + 2\sqrt{q}$, and u is even. Hence for a given q, r_{min} should be $\le (q + 1 - 2\sqrt{q})/2$.
4. l_{max} is typically a small integer (e.g. $l_{max} = 255$).
5. The order $\#E(F_q)$ can be computed by using Schoof's algorithm [37]. Although the basic algorithm is quite inefficient, several dramatic improvements and extensions of this method have been discovered in recent years. Currently, it is feasible to compute orders of elliptic curves over F_p where p is as large as 10^{499}, and orders of elliptic curves over F_{2^m} where m is as large as 1300. Cryptographically suitable elliptic curves over fields as large as $F_{2^{196}}$ can be randomly generated in about 5 hours on a workstation (see [24] and [25]).
6. One technique for selecting an elliptic curve of known order is to use the Weil Theorem which states the following. Let E be an elliptic curve defined over F_q, and let $t = q + 1 - \#E(F_q)$. Let α and β be the complex numbers that satisfy $T^2 - tT + q = (T - \alpha)(T - \beta)$. Then $\#E(F_{q^k}) = q^k + 1 - \alpha^k - \beta^k$ for all $k \ge 1$.

   The Weil Theorem can be used to select a curve over F_{2^m} when m is divisible by a small number l as follows. First select a random elliptic curve E: $y^2 + xy = x^3 + ax^2 + b$, $b \ne 0$, where $a, b \in F_{2^l}$. Note that since l divides m, F_{2^l} is contained in F_{2^m}. Compute $\#E(F_{2^l})$; this can easily be done exhaustively since l is small. Then compute $\#E(F_{2^m})$ using the Weil Theorem with $q = 2^l$ and $k = m/l$.

Section H.4 and Section H.5 presents sample elliptic curves over a 192-bit prime field, a 239-bit prime field, a 256-bit prime field, and the fields $F_{2^{163}}$, $F_{2^{176}}$, $F_{2^{191}}$, $F_{2^{208}}$, $F_{2^{239}}$, $F_{2^{272}}$, $F_{2^{304}}$, $F_{2^{359}}$, $F_{2^{368}}$ and $F_{2^{431}}$ which may be used to ensure the correct implementation of this Standard.

F.3.3. Selecting an Elliptic Curve Verifiably at Random

In order to verify that a given elliptic curve was indeed generated at random, the defining parameters of the elliptic curve are defined to be outputs of the hash function SHA-1 (as specified in ANSI X9.30-1993). The input (SEED) to SHA-1 then serves as proof (under the assumption that SHA-1 cannot be inverted) that the parameters were indeed generated at random. (See Section F.3.4.) The algorithms in this section are used in Section F.3.2.

F.3.3.a. Elliptic curves over F_{2^m}

Input: A field size $q = 2^m$.

Output: A bit string SEED and field elements $a, b \in F_{2^m}$ which define an elliptic curve over F_{2^m}.

Let $t = m$, $s = \lfloor (t-1)/160 \rfloor$, and $h = t - 160 \cdot s$.

1. Choose an arbitrary bit string SEED of bit length at least 160 bits. Let g be the length of SEED in bits.
2. Compute H = SHA-1(SEED), and let b_0 denote the bit string of length h bits obtained by taking the h rightmost bits of H.
3. For i from 1 to s do:

   Compute b_i = SHA-1((SEED + i) mod 2^g).
4. Let b be the field element obtained by the concatenation of $b_0, b_1, \ldots, b_s$ as follows:

   $$b = b_0 \| b_1 \| \ldots \| b_s.$$
5. If $b = 0$, then go to step 1.

6. Let a be an arbitrary element in F_{2^m}.
7. The elliptic curve chosen over F_{2^m} is
$$E : y^2+xy = x^3+ax^2+b.$$
8. Output (SEED, a, b).

F.3.3.b. Elliptic curves over F_p

Input: A prime field size p.

Output: A bit string SEED and field elements $a, b \in F_p$ which define an elliptic curve over F_p.

Let $t = \lceil \log_2 p \rceil$, $s = \lfloor (t-1)/160 \rfloor$, and $h = t - 160 \cdot s$.

1. Choose an arbitrary bit string SEED of bit length at least 160 bits. Let g be the length of SEED in bits.
2. Compute H = SHA-1(SEED), and let c_0 denote the bit string of length h bits obtained by taking the h rightmost bits of H.
3. Let W_0 denote the bit string of length h bits obtained by setting the leftmost bit of c_0 to 0. (This ensures that $r < p$.)
4. For i from 1 to s do:
   Compute W_i = SHA-1((SEED + i) mod 2^g).
5. Let W be the bit string obtained by the concatenation of W_0, W_1,…, W_s as follows:
$$W = W_0 \| W_1 \| ... \| W_s.$$
6. Let $w_1, w_2, \ldots, w_t$ be the bits of W from leftmost to rightmost. Let r be the integer
$$r = \sum_{i=1}^{t} w_i 2^{t-i}.$$
7. Choose integers $a, b \in F_p$ such that $r \bullet b^2 \equiv a^3 \pmod p$. (It is not necessary that a and b be chosen at random. For example, one may choose $a = r$ and $b = r$.)
8. If $4a^3+27b^2 \equiv 0 \pmod p$, then go to step 1.
9. The elliptic curve chosen over F_p is
$$E : y^2 = x^3+ax+b.$$
10. Output (SEED, a, b).

F.3.4. Verifying that an Elliptic Curve was Generated at Random

The technique specified in this section verifies that the defining parameters of an elliptic curve were indeed selected using the method specified in Section F.3.3.

F.3.4.a. Elliptic curves over F_{2^m}

Input: A bit string SEED and a field element $b \in F_{2^m}$.

Output: Acceptance or rejection of the input parameters.

Let $t = m$, $s = \lfloor (t-1)/160 \rfloor$, and $h = t - 160 \cdot s$.

1. Compute H = SHA-1(SEED), and let b_0 denote the bit string of length h bits obtained by taking the h rightmost bits of H.
2. For i from 1 to s do:
   Compute b_i = SHA-1((SEED + i) mod 2^g).
3. Let b' be the field element obtained by the concatenation of $b_0, b_1, \ldots, b_s$ as follows:
$$b' = b_0 \| b_1 \| ... \| b_s.$$
4. If $b = b'$ then accept; otherwise reject.

F.3.4.b. Elliptic curves over F_p

Input: A bit string SEED and field elements $a, b \in F_p$.

Output: Acceptance or rejection of the input parameters.

Let $t = \lceil \log_2 p \rceil$, $s = \lfloor (t-1)/160 \rfloor$, and $h = t - 160 \cdot s$.

1. Compute H = SHA-1(SEED) and let c_0 denote the bit string of length h bits obtained by taking the h rightmost bits of H.
2. Let W_0 denote the bit string of length h bits obtained by setting the leftmost bit of c_0 to 0.
3. For i from 1 to s do:

   Compute W_i = SHA-1((SEED + i) mod 2^g).
4. Let W' be the bit string obtained by the concatenation of $W_0, W_1, \ldots, W_s$ as follows:

   $W' = W_0 \| W_1 \| \ldots \| W_s$.
5. Let $w_1, w_2, \ldots, w_t$ be the bits of W from leftmost to rightmost. Let r' be the integer

   $$r' = \sum_{i=1}^{t} w_i 2^{t-i}.$$
6. If $r' \cdot b^2 \equiv a^3 \pmod{p}$ then accept; otherwise reject.

F.4. Pseudorandom Number Generation

Any implementation of the ECDSA requires the ability to generate random or pseudorandom integers. Such numbers are used to derive a user's private key, d, and a user's per-message secret number k. These randomly or pseudorandomly generated integers are selected to be between 2 and n-2 inclusive, where n is a prime number. If pseudorandom numbers are desired, they shall be generated by the techniques given in this section.

F.4.1. Algorithm Derived from FIPS 186

The algorithm described in this section employs a one-way function $G(t, c)$, where t is 160 bits, c is b bits ($160 \le b \le 512$), and $G(t, c)$ is 160 bits. One way to construct G is via the Secure Hash Algorithm (SHA-1), as defined in ANSI X9.30 Part 2. A second method for constructing G is to use the Data Encryption Algorithm (DEA) as specified in ANSI X9.32. The construction of G by these techniques is described in Sections F.4.1.a and F.4.1.b, respectively.

In the algorithm specified below, a secret b-bit seed-key XKEY is used. If G is constructed via SHA-1 as defined in Section F.4.1.a, then b shall be between 160 and 512. If DEA is used to construct G as defined in Section F.4.1.b, then b shall be equal to 160. The algorithm optionally allows the use of a user provided input.

Input: A prime number n, positive integer l, and integer $b (160 \le b \le 512)$.

Output: l pseudorandom integers $k_1, k_2, \ldots, k_l$ in the interval [1, n - 1].

1. Let $s = \lfloor \log_2 n \rfloor + 1$ and $f = \lceil s/160 \rceil$.
2. Choose a new, secret value for the seed-key, XKEY. (XKEY is of length b bits.)
3. In hexadecimal notation let

   t = 67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0.

   This is the initial value for $H_0 \| H_1 \| H_2 \| H_3 \| H_4$ in SHA-1.
4. For i from 1 to l do the following:
   - 4.1. For j from 1 to f do the following:
     - (a) $XSEED_{i,j}$ = optional user input.
     - (b) XVAL = (XKEY + $XSEED_{i,j}$) mod 2^b.
     - (c) $x_j = G(t, XVAL)$.
     - (d) XKEY = (1 + XKEY + x_j) mod 2^b.
   - 4.2. Set $k_i = ((x_1 \| x_2 \| \ldots \| x_f) \bmod (n-1))$.
   - 4.3. If $k_i = 0$, then set $k_i = n - 1$.

1. Output (k_1, k_2, . . . , k_l).

F.4.1.a. Constructing the Function G from the SHA-1

$G(t,c)$ may be constructed using steps (a)-(e) in Section 3.3 of ANSI X9.30 Part 2. Before executing these steps, $\{H_j\}$ and M_1 must be initialized as follows:

1. Initialize the $\{H_j\}$ by dividing the 160-bit value t into five 32-bit segments as follows:

   $t = t_0 \| t_1 \| t_2 \| t_3 \| t_4$.

   Then $H_j = t_j$ for $j = 0$ through 4.
2. There will be only one message block, M_1, which is initialized as follows:

   $M_1 = c \| 0^{512-b}$.

   (The first b bits of M_1 contain c, and the remaining (512-b) bits are set to zero.)

   Then steps (a) through (e) of Section 3.3 of ANSI X9.30 Part 2 are executed, and $G(t,c)$ is the 160-bit string represented by the five words:

   $H_0 \| H_1 \| H_2 \| H_3 \| H_4$

   at the end of step (e).

F.4.1.b. Constructing the Function G from the DEA

$G(t, c)$ may be constructed using the DEA (Data Encryption Algorithm) as specified in ANSI X3.92.

Let $a \oplus b$ denote the bitwise exclusive-or of bit strings a and b, and let $a \| b$ denote the concatenation of bit strings. If $b1$ is a 32-bit string, then $b1'$ denotes the 24 least significant bits of $b1$.

In the following, $DEA_K(A)$ represents ordinary DEA encryption of the 64-bit block A using the 56-bit key K. Now suppose t and c are each 160 bits. To compute $G(t,c)$:

Step 1: Write:

$t = t_1 \| t_2 \| t_3 \| t_4 \| t_5$

$c = c_1 \| c_2 \| c_3 \| c_4 \| c_5$.

In the above, t_i and c_i are each 32 bits in length.

Step 2: For i from 1 to 5 do:

$x_i = t_i \oplus c_i$.

Step 3: For i from 1 to 5 do:

$b1 = c_{((i+3) \bmod 5) + 1}$

$b2 = c_{((i+2) \bmod 5) + 1}$

$a1 = x_i$

$a2 = x_{(i \bmod 5) + 1} \oplus x_{((i+3) \bmod 5)+1}$

$y_{i,1} \| y_{i,2} = DEA_{b1' \| b2}(a1 \| a2)$ where $y_{i,1}$ and $y_{i,2}$ are each 32 bits in length.

Step 4: For i from 1 to 5 do:

$z_i = y_{i,1} \oplus y_{((i+1) \bmod 5)+1,2} \oplus y_{((i+2) \bmod 5)+1,1}$.

Step 5: Let $G(t,c) = z_1 \| z_2 \| z_3 \| z_4 \| z_5$.

F.4.2 Algorithm from ANSI X9.17

The technique in this section is from Appendix C of ANSI X9.17, "Financial Institution Key Management (Wholesale)".

Let $ede^*X(Y)$ represent the DEA multiple encryption of Y under the key $^*X$. Let $^*K$ be a DEA key pair reserved only for the generation of pseudorandom numbers, let V be a 64-bit seed value which is also kept secret, and let $\oplus$ be the exclusive-or operator. Let DT be a date/time vector which is updated on each iteration. I is an intermediate value. A 64-bit vector R is generated as follows:

$I = ede * K(DT)$

$R = ede * K(I \oplus V)$

and a new V is generated by $V = ede * K(R \oplus I)$.

Successive values of R may be concatenated to produce a pseudorandom number of the desired length.

Appendix G Informative Number-Theoretic Algorithms
[Informative]

G.1. Finite Fields and Modular Arithmetic

G.1.1. Exponentiation in a Finite Field

If a is a positive integer and g is an element of the field F_q, then *exponentiation* is the process of computing g^a. Exponentiation can be performed efficiently by the *binary method* outlined below. The algorithm is used in Sections G.1.2 and G.1.4.

Input: A positive integer a, a field F_q, and a field element g.

Output: g^a.

1. Set $e = a \bmod (q - 1)$. If $e = 0$, then output 1.
2. Let $e = e_r e_{r-1} \ldots e_1 e_0$ be the binary representation of e, where the most significant bit e_r of e is 1.
3. Set $x = g$.
4. For i from $r - 1$ downto 0 do
   - 4.1. Set $x = x^2$.
   - 4.2. If $e_i = 1$, then set $x = gx$.
5. Output x.

There are several variations of this method which can be used to speed up the computations. One such method which requires some precomputations is described in [21]. See also Knuth [21, pp. 441-466].

G.1.2. Inversion in a Finite Field

If $g \neq 0$ is an element of the field F_q, then the *inverse* g^{-1} is the field element c such that $gc = 1$. The inverse can be found efficiently by exponentiation since $c = g^{q-2}$. Note that if q is prime and g is an integer satisfying $1 \leq g \leq q - 1$, then g^{-1} is the integer c, $1 \leq g \leq q - 1$, such that $gc \equiv 1 \pmod{q}$. The algorithm is used in Sections 5.3.3 and 5.4.2.

Input: A field F_q, and a non-zero element $g \in F_q$.

Output: The inverse g^{-1}.

1. Compute $c = g^{q-2}$ (see Section G.1.1).
2. Output c.

An even more efficient method is the extended Euclidean Algorithm [21, p. 325].

G.1.3. Generating Lucas Sequences

Let P and Q be nonzero integers. The *Lucas sequences* U_k and V_k for P, Q are defined by

$U_0 = 0$, $U_1 = 1$, and $U_k = PU_{k-1} - QU_{k-2}$ for $k \geq 2$.

$V_0 = 2$, $V_1 = P$, and $V_k = PV_{k-1} - QV_{k-2}$ for $k \geq 2$.

This recursion is adequate for computing U_k and V_k for small values of k. The following algorithm can be used to efficiently compute U_k and V_k modulo an odd prime p for large values of k. The algorithm is used in Section G.1.4.

Input: An odd prime p, integers P and Q, and a positive integer k.

Output: $U_k \bmod p$ and $V_k \bmod p$

1. Set $\Delta = P^2 - 4Q$.
2. Let $k = k_r k_{r-1} \ldots k_1 k_0$ be the binary representation of k, where the leftmost bit k_r of k is 1.
3. Set $U = 1$, $V = P$.
4. For i from $r - 1$ downto 0 do
   - 4.1. Set $(U,V) = (UV \bmod p, \frac{(V^2 + \Delta U^2)}{2} \bmod p)$.
   - 4.2. If $k_i = 1$ then set $(U,V) = (\frac{(PU + V)}{2} \bmod p, \frac{(PV + \Delta U)}{2} \bmod p)$.
5. Output U and V.

G.1.4. Finding Square Roots Modulo a Prime

Let p be an odd prime, and let g be an integer with $0 \le g < p$. A square root (mod p) of g is an integer y with $0 \le y < p$ and

$$y^2 \equiv g \pmod{p}.$$

If $g = 0$, then there is one square root (mod p), namely $y = 0$. If $g \neq 0$, then g has either 0 or 2 square roots (mod p). If y is one square root, then the other is $p - y$.

The following algorithm determine whether g has square roots (mod p) and, if so, compute one. The algorithm is used in Sections 4.4.1.a and G.3.1.

Input: An odd prime p, and an integer g with $0 < g < p$.

Output: A square root (mod p) of g if one exists; otherwise, the message "no square roots exist."

ALGORITHM 1: *for* $p \equiv 3 \pmod 4$, *that is* $p = 4u + 3$ *for some positive integer* u.

1. Compute $y = g^{u+1} \pmod{p}$ via Section G.1.1.
2. Compute $z = y^2 \bmod p$.
3. If $z = g$, then output y. Otherwise output the message "no square roots exist."

ALGORITHM 2: *for* $p \equiv 5 \pmod 8$, *that is* $p = 8u + 5$ *for some positive integer* u.

1. Compute $\gamma = (2g)^u \bmod p$ via Section G.1.1.
2. Compute $i = 2g\gamma^2 \bmod p$.
3. Compute $y = g\gamma(i - 1) \bmod p$.
4. Compute $z = y^2 \bmod p$.
5. If $z = g$, then output y. Otherwise output the message "no square roots exist."

ALGORITHM 3: *for* $p \equiv 1 \pmod 4$, *that is* $p = 4u + 1$ *for some positive integer* u.

1. Set $Q \leftarrow g$.
2. Generate random P with $0 \le P < p$.
3. Using Section G.1.3, compute the Lucas sequence elements
$$U = U_{2k+1} \bmod p, \quad V = V_{2k+1} \bmod p$$
4. If $V^2 \equiv 4Q \pmod p$ then output $y = V/2 \bmod p$ and stop.
5. If $U \not\equiv \pm 1 \pmod p$ then output the message "no square roots exist" and stop.
6. Go to Step 2.

G.1.5. Trace and Half-Trace Functions

If α is an element of F_{2^m}, the *trace* of α is:

$$Tr(\alpha) = \alpha + \alpha^2 + \alpha^{2^2} + ... + \alpha^{2^{m-1}}.$$

The value of $Tr(\alpha)$ is 0 for half the elements of F_{2^m}, and 1 for the other half.

The trace can be computed as follows. The methods are used in Section G.1.6.

Normal basis representation used for elements of F_{2^m}:

If α has representation $(\alpha_0 \alpha_1 ... \alpha_{m-1})$, then

$$Tr(\alpha) = \alpha_0 \oplus \alpha_1 \oplus ... \oplus \alpha_{m-1}.$$

Polynomial basis representation used for elements of F_{2^m}:

1. Set $T = \alpha$.
2. For i from 1 to $m - 1$ do
   2.1. $T = T^2 + \alpha$.
3. Output T.

If m is odd, the *half-trace* of α is

$\alpha + \alpha^{2^2} + \alpha^{2^4} + ... + \alpha^{2^{m-1}}$.

If F_{2^m} is represented by a polynomial basis, the half-trace can be computed efficiently as follows. The method is used in Section G.1.6.

1. Set $T = \alpha$.
2. For i from 1 to $(m - 1)/2$ do
   2.1. $T = T^2$.
   2.2. $T = T^2 + \alpha$.
3. Output T.

G.1.6. Solving Quadratic Equations over F_{2^m}

If β is an element of F_{2^m}, then the equation:

$z^2 + z = \beta$

has $2 - 2T$ solutions over F_{2^m}, where $T = Tr(\beta)$. Thus, there are either 0 or 2 solutions.

If $\beta = 0$, then the solutions are 0 and 1. If $\beta \neq 0$ and z is a solution, then the other solution is $z + 1$.

The following algorithms determine whether a solution z exists for a given β, and if so, computes one. The algorithms are used in point compression (see Section 4.4.1.b) and in Section G.3.1.

Input: A field F_{2^m} along with a basis for representing its elements; and an element $\beta \neq 0$.

Output: An element z for which $z^2 + z = \beta$ if any exist; otherwise the message "no solutions exist".

ALGORITHM 1: *for normal basis representation*.

1. Let $(\beta_0 \beta_1 ... \beta_{m-1})$ be the representation of β.
2. Set $z_0 = 0$.
3. For i from 1 to m-1 do
   3.1. Set $z_i = z_{i-1} \oplus \beta_i$.
4. Set $z = (z_0 z_1 ... z_{m-1})$.
5. Compute $\gamma = z^2 + z$.
6. If $\gamma = \beta$ then output z. Otherwise output the message "no solutions exist".

ALGORITHM 2: *for polynomial basis representation, with m odd*.

1. Compute z = half-trace of β via Section G.1.5.
2. Compute $\gamma = z^2 + z$.
3. If $\gamma = \beta$ then output z. Otherwise output the message "no solutions exist".

ALGORITHM 3: *works in any polynomial basis*.

1. Choose a random $\tau \in F_{2^m}$.
2. Set $z = 0$ and $w = \beta$.
3. For i from 1 to m - 1 do
   3.1. Set $z = z^2 + w^2\tau$.
   3.2. Set $w = w^2 + \beta$.
4. If $w \neq 0$ then output the message "no solutions exist" and stop.
5. Compute $\gamma = z^2 + z$.
6. If $\gamma = 0$ then go to Step 1.
7. Output z.

G.1.7. Checking the Order of an Integer Modulo a Prime

Let p be a prime and let g satisfy $1 < g < p$. The *order* of g modulo p is the smallest positive integer k such that $g^k \equiv 1 \pmod{p}$. The following algorithm tests whether or not g has order k modulo p. The algorithm is used in Section G.1.8.

Input: A prime p, a positive integer k, and an integer g with $1 < g < p$.

Output: "True" if g has order k modulo p, and "False" otherwise.

1. Determine the prime divisors of k.
2. If g^k 1 (mod p), then output "False" and stop.
3. For each prime l dividing k do
   - 3.1. If $g^{k/l} \equiv 1 \pmod{p}$, then output "False" and stop.
4. Output "True".

G.1.8. Checking the Existence of an Optimal Normal Basis

The following algorithm determines whether an optimal normal basis of Type I or Type II (or both) exists for F_{2^m}.

Input: A positive integer $m > 1$.

Output: A message "Type I only", "Type II only", "Both Types", or "Neither Type".

1. Set T_1 = "False" and T_2 = "False".
2. If $m \equiv 2$ or 4 (mod 8) and $p = m + 1$ is prime, then
   - 2.1. Test (using the technique defined in Section G.1.7) whether 2 has order p - 1 modulo p. Let T_1 be the output of this test ("True" or "False").
3. If $m \equiv 1$ or 2 (mod 4) and $p = 2m + 1$ is prime, then
   - 3.1. Test (using the technique defined in Section G.1.7) whether 2 has order p - 1 modulo p. Let T_2 be the output of this test ("True" or "False").
4. If $m \equiv 3 \pmod{4}$ and $p = 2m + 1$ is prime, then:
   - 4.1. Test (using the technique defined in Section G.1.7) whether 2 has order $(p - 1)/2$ modulo p. Let T_2 be the output of this test ("True" or "False").
5. If T_1 ="True" and T_2 = "True" then output "Both Types".
   - 5.1. If T_1 ="True" and T_2 = "False" then output "Type I only".
   - 5.2. If T_1 ="False" and T_2 = "True" then output "Type II only".
   - 5.3. If T_1 ="False" and T_2 = "False" then output "Neither Type".

G.2. Polynomials over a Finite Field

G.2.1. GCD's over a Finite Field

If $f(t)$ and $g(t) \neq 0$ are two polynomials with coefficients in the field F_q, then there is a unique monic polynomial $d(t)$ of largest degree which divides both $f(t)$ and $g(t)$. The polynomial $d(t)$ is called the *greatest common divisor* or *gcd* of $f(t)$ and $g(t)$. The following algorithm (the Euclidean algorithm) computes the gcd of two polynomials. The algorithm is used in Section G.2.2.

Input: A finite field F_q and two polynomials $f(t)$, $g(t) \neq 0$ over F_q.

Output: $d(t) = \gcd(f(t), g(t))$.

1. Set $a(t) = f(t)$, $b(t) = g(t)$.
2. While $b(t) \neq 0$
   - 2.1. Set $c(t)$ = the remainder when $a(t)$ is divided by $b(t)$.
   - 2.2. Set $a(t) = b(t)$.
   - 2.3. Set $b(t) = c(t)$.
3. Let α be the leading coefficient of $a(t)$ and output $\alpha^{-1}a(t)$.

G.2.2. Finding a Root in F_{2^m} of an Irreducible Binary Polynomial

If $f(t)$ is an irreducible polynomial (mod 2) of degree m, then $f(t)$ has m distinct roots in the field F_{2^m}. A random root can be found efficiently using the following algorithm. The algorithm is used in Section G.2.3.

Input: An irreducible polynomial $f(t)$ of degree m over F_2, and a field F_{2^m}.

Output: A random root of $f(t)$ in F_{2^m}.

1. Set $g(t) = f(t)$.
2. While $\deg(g) > 1$
   - 2.1. Choose random $u \in F_{2^m}$.
   - 2.2. Set $c(t) = ut$.
   - 2.3. For i from 1 to m - 1 do
     - 2.3.1. $c(t) = (c(t)^2 + ut) \bmod g(t)$.
   - 2.4. Set $h(t) = \gcd(c(t), g(t))$.
   - 2.5. If $h(t)$ is constant or $\deg(g) = \deg(h)$, then go to step 2.1.
   - 2.6. If $2\deg(h) > \deg(g)$, then set $g(t) = g(t) / h(t)$; else $g(t) = h(t)$.
3. Output $g(0)$.

G.2.3. Change of Basis

Given a field F_{2^m} and two (polynomial or normal) bases B_1 and B_2 for the field over F_2, the following algorithm allows conversion between bases B_1 and B_2.

1. Let $f(t)$ be the field polynomial of B_2. That is,
   - 1.1. If B_2 is a polynomial basis, let $f(t)$ be the (irreducible) reduction polynomial of degree m over F_2.
   - 1.2. If B_2 is a Type I optimal normal basis (see Section G.1.8), let:
     $$f(t) = t^m + t^{m-1} + t^{m-2} + \ldots + t + 1.$$
   - 1.3. If B_2 is a Type II optimal normal basis (see Section G.1.8), let:
     $$f(t) = \sum_{\substack{0 \le j \le m \\ m-j \,\underline{\pi}\, m+j}} t^j$$
     where the notation $a \,\underline{\pi}\, b$ means that in the binary representations
     $$a = \sum u_i 2^i,\ b = \sum w_i 2^i,$$
     we have $u_i \le w_i$ for all i. (These polynomials can also be calculated using the recursion from Section 4.1.4.a.)
2. Let γ be a root of $f(t)$ computed with respect to B_1. (γ can be computed using the technique defined in Section G.2.2.)
3. Let Γ be the matrix:
$$\Gamma = \begin{bmatrix} \gamma_{0,0} & \gamma_{0,1} & \Lambda & \gamma_{0,m-1} \\ \gamma_{1,0} & \gamma_{1,1} & \Lambda & \gamma_{1,m-1} \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & \mathrm{M} \\ \gamma_{m-1,0} & \gamma_{m-1,1} & \mathrm{K} & \gamma_{m-1,m-1} \end{bmatrix},$$
where the entries $\gamma_{i,j}$ are defined as follows:
   - 3.1. If B_2 is a polynomial basis, then:
     $$1 = (\gamma_{0,0}\ \gamma_{0,1} \ldots \gamma_{0,m-1})$$
     $$\gamma = (\gamma_{1,0}\ \gamma_{1,1} \ldots \gamma_{1,m-1})$$
     $$\gamma^2 = (\gamma_{2,0}\ \gamma_{2,1} \ldots \gamma_{2,m-1})$$
     $$\bullet$$

•

•

$\gamma^{m-1} = (\gamma_{m-1,0}\ \gamma_{m-1,1}\ \ldots\ \gamma_{m-1,m-1})$

with respect to B_1. (The entries $\gamma_{i,j}$ are computed by repeated multiplication by γ.)

3.2. If B_2 is an optimal normal basis, then:

$\gamma = (\gamma_{0,0}\ \gamma_{0,1}\ \ldots\ \gamma_{0,m-1})$
$\gamma^2 = (\gamma_{1,0}\ \gamma_{1,1}\ \ldots\ \gamma_{1,m-1})$
$\gamma^4 = (\gamma_{2,0}\ \gamma_{2,1}\ \ldots\ \gamma_{2,m-1})$

•

•

•

$\gamma^{2^{m-1}} = (\gamma_{m-1,0}\ \gamma_{m-1,1}\ \ldots\ \gamma_{m-1,m-1})$

with respect to B_1. (The entries $\gamma_{i,j}$ are computed by repeated squaring of γ.)

4. If an element has representation $(\beta_0\beta_1\ldots\beta_{m-1})$ with respect to B_2, then its representation with respect to B_1 is

$$(\alpha_0\alpha_1\ldots\alpha_{m-1}) = (\beta_0\beta_1\ldots\beta_{m-1})\,\Gamma.$$

If an element has representation $(\alpha_0\alpha_1\ldots\alpha_{m-1})$ with respect to B_1, then its representation with respect to B_2 is

$$(\beta_0\beta_1\ldots\beta_{m-1}) = (\alpha_0\alpha_1\ldots\alpha_{m-1})\,\Gamma^{-1}.$$

Example : Suppose that B_1 is the polynomial basis (mod $t^4 + t + 1$), and B_2 is the Type I optimal normal basis for F_{2^4}.

Then: $f(t) = t^4 + t^3 + t^2 + t + 1$, and a root is given by $\gamma = (1100)$ with respect to B_1. Then:

$\gamma = (1100)$

$\gamma^2 = (1111)$

$\gamma^4 = (1010)$

$\gamma^8 = (1000)$,

so that:

$$\Gamma = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

and:

$$\Gamma^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

If $\lambda = (1001)$ with respect to B_2, then its representation with respect to B_1 is:

$(0100) = (1001)\,\Gamma$.

If $\lambda = (1011)$ with respect to B_1, then its representation with respect to B_2 is:

$(1101) = (1011)\ \Gamma^{-1}$.

G.2.4. Checking Polynomials over F_2 for Irreducibility

If $f(x)$ is a polynomial with coefficients in the field F_2, then $f(x)$ can be tested efficiently for irreducibility using the following algorithm. The algorithm is used in Section 5.1.2.b.

Input: A polynomial $f(x)$ with coefficients in F_2.

Output: The message "True" if $f(x)$ is irreducible over F_2; the message "False" otherwise.

1. Set d = degree of $f(x)$.
2. Set $u(x) = x$.
3. For i from 1 to $\lfloor d/2 \rfloor$ do
   - 3.1. Set $u(x) = u(x)^2 \bmod f(x)$.
   - 3.2. Set $g(x) = \gcd(u(x) + x, f(x))$.
   - 3.3. If $g(x) \neq 1$ then output "False" and stop.
4. Output "True".

G.3. Elliptic Curve Algorithms

G.3.1. Finding a Point on an Elliptic Curve

The following algorithms provide an efficient method for finding a random point (other than $\mathcal{O}$) on a given elliptic curve over a finite field. These algorithms are used in Sections F.3.1 and F.3.2.

Case I: Curves over F_p

Input: A prime p and the parameters a and b of an elliptic curve E over F_p.

Output: A randomly generated point (other than $\mathcal{O}$) on E.

1. Choose a random integer x with $0 \leq x < p$.
2. Set $\alpha = x^3 + ax + b \bmod p$.
3. If $\alpha = 0$ then output $(x, 0)$ and stop.
4. Apply the appropriate algorithm from Section G.1.4 to look for a square root (mod p) of α.
5. If the output of Step 4 is "no square roots exist," then go to Step 1. Otherwise the output of Step 4 is an integer y with $0 < y < p$ such that $y^2 \equiv \alpha \pmod p$.
6. Output (x, y).

Case II: Curves over F_{2^m}**.**

Input: A field F_{2^m} and the parameters a and b of an elliptic curve E over F_{2^m}.

Output: A randomly generated point (other than $\mathcal{O}$) on E.

1. Choose a random element x in F_{2^m}.
2. If $x = 0$, then output $(0, b^{2^{m-1}})$ and stop.
3. Set $\alpha = x^3 + ax^2 + b$.
4. If $\alpha = 0$, then output $(x, 0)$ and stop.
5. Set $\beta = x^{-2}\alpha$.
6. Apply the appropriate algorithm from Section G.1.6 to look for an element z for which $z^2 + z = \beta$.
7. If the output of Step 6 is "no solutions exist," then go to Step 1. Otherwise the output of Step 6 is a solution z.
8. Set $y = xz$.
9. Output (x, y).

G.3.2. Computing a Multiple of an Elliptic Curve Point

If k is a positive integer and P is an elliptic curve point, then kP is the point obtained by adding together k copies of P. This computation can be performed efficiently by the *addition-subtraction method* outlined below. These algorithms are used, for example, in Sections 5.1.1, 5.1.2, 5.3, and 5.4.

Input: A positive integer k and an elliptic curve point P.

Output: The elliptic curve point kP.

1. Set $e = k \bmod n$, where n is the order of P. (If n is unknown, then set $e = k$ instead.)
2. Let $h_r h_{r-1} ... h_1 h_0$ be the binary representation of $3e$, where the leftmost bit h_r is 1.
3. Let $e_r e_{r-1} ... e_1 e_0$ be the binary representation of e.
4. Set $Q = P$.
5. For i from $r - 1$ down to 1 do
   - 5.1. Set $Q = 2Q$.
   - 5.2. If $h_i = 1$ and $e_i = 0$, then set $Q = Q + P$.
   - 5.3. If $h_i = 0$ and $e_i = 1$, then set $Q = Q - P$.
6. Output Q.

Note: To subtract the point (x, y), just add the point $(x, -y)$ (for the field F_p) or $(x, x + y)$ (for the field F_{2^m}).

There are several variations of this method which can be used to speed up the computations. One such method which requires some precomputations is described in [10]. See also Knuth [21, pages 441-466].

Appendix H Examples of ECDSA and Sample Curves
[Informative]

This appendix contains 5 parts.

- Section H.1 presents examples of data conversion methods.
  - Section H.2 presents 2 examples of ECDSA over the field F_{2^m}. [3]
  - Section H.3 presents 2 examples of ECDSA over the field F_p , where p is odd prime;
- Section H.4 presents sample elliptic curves over the field F_{2^m} with system parameters for m = 163, 176, 191, 208, 239, 272, 304, 359, 368 and 431.
- Section H.5 presents sample elliptic curves over field F_p with system parameters for 192-bit, 239-bit, and 256-bit primes.

The sample curves in Sections H.4 and H.5 may be used in an implementation of this Standard.

H.1. Examples of Data Conversion Methods

The following are examples of the data conversion techniques that shall be used in this Standard (See Figure 1).

Example of Integer-to-Octet-String Conversion. (See Section 4.2.1.)

Input: x = 123456789, k=4

Output: M= 075BCD15

Example of Octet-String-to-Integer Conversion. (See Section 4.2.2.)

Input: M= 0003ABF1CD

Output: x = 61600205

Example of Field-Element-to-Octet-String Conversion. (See Section 4.3.1.)

(i) **Input:** α = 94311, q =104729 (an odd prime).

Output: S = 017067 (l=3).

(ii) **Input:** α = 1101101101110111100110111111110110111110001 , q=2^{41}.

Output: S = 01B6EF37EDF1 (l=6).

Example of Octet-String-to-Field-Element Conversion. (See Section 4.3.2.)

(i) **Input:** S = 01E74E (l=3), q = 224737 (an odd prime).

Output: α = 124750.

(ii) **Input:** S = 0117B2939ACC (l=6), q=2^{41}.

Output: α= 100010111101100101001001110011010110011100.

Example of Field-Element-to-Integer Conversion. (See Section 4.3.3.)

(i) **Input:** α = 136567, q = 287117, (an odd prime).

Output: x = 136567.

(ii) **Input:** α = 111111110010000100111100001100111101011110 , q=2^{41}.

Output: x = 2191548508078.

---

[3] The curves in Sections H.4.1, H.4.3, H.4.5, H.4.8 and H.4.10 were generated using a software package by R. Lercier and F. Morain that is described in [25].

Example of Point-to-Octet-String Conversion. (See Section 4.4.2.a.)

(i) **Input:**

p = 6277101735386680763835789423207666416083908700390324961279

The curve is E: $y^2=x^3+ax+b$ where

a = 6277101735386680763835789423207666416083908700390324961276

b = 245515554600894381774029391519745178476910805816119123 8065

The point is $P=(x_1, y_1)$, where

x_1 = 602046282375688656758213480587526111916698976636884684818

y_1 = 174050332293622031404857552280219410364023488927386650641

Output: (compressed form)

PO = 03 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012

Output: (uncompressed form)

PO = 04 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811

Output: (hybrid form)

PO = 07 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811

(ii) **Input:** $q=2^{191}$

The field $F_{2^{191}}$ is generated by the irreducible polynomial

f = 80000000 00000000 00000000 00000000 00000000 00000201

The curve is E: $y^2+xy=x^3+ax^2+b$ over $F_{2^{191}}$, where

a = 2866537B 67675263 6A68F565 54E12640 276B649E F7526267

b = 2E45EF57 1F00786F 67B0081B 9495A3D9 5462F5DE 0AA185EC

The point is $P=(x_1, y_1)$, where

x_1 = 1101101011001111011010111110001010001000110010000001101111110011100010011110010100110011101011110110010000110101001110000110110100100010011011111110010110010000010010101110000110101010000001101

y_1 = 1110110010110111110011100110100001100111011001111111001010111110001100110010100100110010111001110000111010100010001000101110010100010010000011000111010101000001110111110011000000000001100011111011

Output: (compressed form)

PO = 02 36B3DAF8 A23206F9 C4F299D7 B21A9C36 9137F2C8 4AE1AA0D

Output: (uncompressed form)

PO = 04 36B3DAF8 A23206F9 C4F299D7 B21A9C36 9137F2C8 4AE1AA0D 765BE734 33B3F95E 332932E7 0EA245CA 2418EA0E F98018FB

Output: (hybrid)

PO = 06 36B3DAF8 A23206F9 C4F299D7 B21A9C36 9137F2C8 4AE1AA0D 765BE734 33B3F95E 332932E7 0EA245CA 2418EA0E F98018FB

Example of Octet-String-to-Point Conversion. (See Section 4.4.2.b.)

(i) **Input:**

p = 6277101735386680763835789423207666416083908700390324961279

The curve is E: $y^2=x^3+ax+b$ where

a = 6277101735386680763835789423207666416083908700390324961276

b = 500540239228939020355206947077111708486189930780145699 0547

The octet string is

03 EEA2BAE7 E1497842 F2DE7769 CFE9C989 C072AD69 6F48034A

Output: The point is $P=(x_1, y_1)$, where

x_1 = 5851329466723574623122023978072381191095567081251774399306

y_1 = 2487701625881228691269808880535093938601070911264778280469

(ii) **Input:** $q=2^{191}$

The field $F_{2^{191}}$ is generated by the irreducible polynomial

f = 80000000 00000000 00000000 00000000 00000000 00000201

The curve is E: $y^2+xy=x^3+ax^2+b$ over $F_{2^{191}}$, where

a = 40102877 4D7777C7 B7666D13 66EA4320 71274F89 FF01E718

b = 0620048D 28BCBD03 B6249C99 182B7C8C D19700C3 62C46A01

The octet string is

PO = 02 3809B2B7 CC1B28CC 5A87926A AD83FD28 789E81E2 C9E3BF10

Output: The point is $P=(x_1, y_1)$, where

x_1 = 1110000000100110110010101101111100110000011011001010001100110001011010100000111100100100110101010101101100000011111111010010100001111000100111101000000111100010110010010001111000111011111100010000

y_1 = 1011101000011010000111000011001100010011011010001010011110011101101111100000001011101100000110110010010000100111010000111110000111001111001101111101011101100010000011011111101011001101000101 0

H.2. Examples of ECDSA over the Field F_{2^m}

H.2.1. An Example With m = 191 (trinomial basis)

a. Elliptic Curve Parameter Setup

1. The field $F_{2^{191}}$ is generated by the irreducible polynomial

f = 80000000 00000000 00000000 00000000 00000000 00000201

2. The curve is $E : y^2+xy=x^3+ax^2+b$ over $F_{2^{191}}$, where

seed = 4E13CA54 2744D696 E6768756 1517552F 279A8C84

a = 2866537B 67675263 6A68F565 54E12640 276B649E F7526267

b = 2E45EF57 1F00786F 67B0081B 9495A3D9 5462F5DE 0AA185EC

3. Generating point P (without point compression)

04 36B3DAF8 A23206F9 C4F299D7 B21A9C36 9137F2C8 4AE1AA0D 765BE734 33B3F95E 332932E7 0EA245CA 2418EA0E F98018FB

P has prime order

n = 1569275433846670190958947355803350458831205595451630533029

h = 2

b. Key Generation

d = 1275552191113212300012030439187146164646146646467494947799

$Q = dP = (x_Q, y_Q)$ (without point compression)

04 5DE37E75 6BD55D72 E3768CB3 96FFEB96 2614DEA4 CE28A2E7 55C0E0E0 2F5FB132 CAF416EF 85B229BB B8E13520 03125BA1

c. Signature Generation

M = "abc"

1. Message digesting

SHA-1 is applied to M to get

e = SHA-1(M)= 968236873715988614170569073515315707566766479517

2. Elliptic curve computation

a. select a k in the interval [2,n-2]

k = 1542725565216523985789236956265265265235675811949404040041

b. compute $R = kP = (x_1, y_1)$

x_1 = 438E5A11 FB55E4C6 5471DCD4 9E266142 A3BDF2BF 9D5772D5

y_1 = 2AD603A0 5BD1D177 649F9167 E6F475B7 E2FF590C 85AF15DA

c. convert x_1 to an integer $\bar{x}_1$

$\bar{x}_1$ = 1656469817011541734314669640730254878828443186986697061077

d. set $r = \bar{x}_1 \bmod n$.

r = 871943831648715433557222849269044199972375915350665280048

e. $r \neq 0$, OK.

3. Modular computation

a. compute $s = k^{-1}(e + dr) \bmod n$.

s = 308992691965804947361541664549085895292153777025772063598

b. $s \neq 0$, OK.

4. Signature formatting

The signature is the two integers r and s.

r = 871943831648715433557222849269044199972375915350665280048

s = 308992691965804947361541664549085895292153777025772063598

d. Signature verification

1. Message digesting

SHA-1 is applied to M' to get

e=SHA-1(M')= 968236873715988614170569073515315707566766479517

2. Elliptic curve computation

a. r' is in interval [1, n-1], OK

b. s' is in interval [1, n-1], OK.

c. compute $c = (s')^{-1} \bmod n$

c = 952933666850866331568782284754801289889992082635386177703

d. compute $u_1 = ec \bmod n$ and $u_2 = r'c \bmod n$

u_1 = 1248886407154707854022434516084062503301792374360994400066

u_2 = 5270173809775340121682224660161998496119711416527534641 54

e. compute $(x_1, y_1) = u_1P + u_2Q$

u_1P = 1A045B0C 26AF1735 9163E9B2 BF1AA57C 5475C320 78ABE159 53ECA58F AE7A4958 783E8173 CF1CA173 EAC47049 DCA02345

u_2Q = 015CF19F E8485BED 8520CA06 BD7FA967 A2CE0B30 4FFCF0F5 314770FA 4484962A EC673905 4A6652BC 07607D93 CAC79921

$u_1P + u_2Q = (x_1, y_1)$

x_1 = 438E5A11 FB55E4C6 5471DCD4 9E266142 A3BDF2BF 9D5772D5

y_1 = 2AD603A0 5BD1D177 649F9167 E6F475B7 E2FF590C 85AF15DA

3. Signature check

a. convert x_1 to an integer $\bar{x}_1$

$\bar{x}_1$ = 165646981701154173431466964073025487882844318698669706 1077

b. compute $v = \bar{x}_1 \bmod n$.

v = 871943831648715433557222849269044199972375915350665280 48

c. $v = r'$. OK

H.2.2. An Example With $m = 239$ (trinomial basis)

a. Elliptic Curve Parameter Setup

1. The field $F_{2^{239}}$ is generated by the irreducible polynomial

f = 8000 00000000 00000000 00000000 00000000 00000000 00000010 00000001

2. The curve is $E : y^2+xy=x^3+ax^2+b$ over $F_{2^{239}}$, where

seed = D34B9A4D 696E6768 75615175 CA71B920 BFEFB05D

a = 3201 0857077C 5431123A 46B80890 6756F543 423E8D27 87757812 5778AC76

b = 7904 08F2EEDA F392B012 EDEFB339 2F30F432 7C0CA3F3 1FC383C4 22AA8C16

3. Generating point P (without point compression)

04 57927098 FA932E7C 0A96D3FD 5B706EF7 E5F5C156
E16B7E7C 86038552 E91D61D8 EE5077C3 3FECF6F1 A16B268D
E469C3C7 744EA9A9 71649FC7 A9616305

P has prime order

n = 2208558830972980411979121875928648145578869937767132309367150412074117 83

h = 4

b. Key Generation

d = 1456427555219115346513212300075341203043918714616464614664664674949479 90

$Q = dP = (x_Q, y_Q)$ (without point compression)

04 5894609C CECF9A92 533F630D E713A958 E96C97CC
B8F5ABB5 A688A238 DEED6DC2 D9D0C94E BFB7D526 BA6A6176
4175B99C B6011E20 47F9F067 293F57F5

c. Signature Generation

M = "abc"

1. Message digesting

SHA-1 is applied to *M* to get

e = SHA-1(M) = 968236873715988614170569073515315707566766479517

2. Elliptic curve computation

a. select a *k* in the interval [2, *n*-2]

k = 1712787255652165239672857892369562652652652356758119494040040041670216363

b. compute $R = kP = (x_1, y_1)$

x_1 = 6321 0D71EF6C 10157C0D 1053DFF9 3EB8F028 1E3F9DA2
DEB377A8 1BDAE8D5

y_1 = 5EAF D217370E 12036519 CAD381A1 FC38234F 61870DB2
2C1E410A C1F183F0

c. convert x_1 to an integer $\bar{x}_1$

$\bar{x}_1$ = 6841639825023137355787549028176290563024791328169814824526010005446269 01

d. set $r = \bar{x}_1 \bmod n$.

r = 2159633321041961198501834003903461262881815148684178964245587692239155 2

e. $r \neq 0$, OK.

3. Modular computation

a. compute $s = k^{-1}(e + dr) \bmod n$.

s = 1970303740007316867383349976549972270528498040721988 19

102649413465737174

b. $s \neq 0$, OK.

4. Signature formatting

The signature is the two integers r and s.

r = 215963332104196119850183400390346126288181514868417896
42455876922391552

s = 197030374000731686738334997654997227052849804072198819
102649413465737174

d. Signature verification

1. Message digesting

SHA-1 is applied to M' to get

e=SHA-1(M')=968236873715988614170569073515315707566766479517

2. Elliptic curve computation

a. r' is in interval [1, n-1], OK

b. s' is in interval [1, n-1], OK.

c. compute $c=(s')^{-1} \bmod n$

c = 431396620921664668890077637965697612042893607943599260
03383145535744433

d. compute $u_1 = ec \bmod n$ and $u_2 = r'c \bmod n$

u_1 = 105375096144033333985559550644017212889091653305446724
555949472922658998

u_2 = 215828469521640156896840216715465581571744240077746044
580128914744769962

e. compute $(x_1, y_1) = u_1P+u_2Q$

u_1P = 12C9 F6F4C153 014AD6E5 04B3036B B47FFD7B D42B820A
00F84CA8 C5C89FCA

78EA 1205C486 3D0CA5DE 16FF6324 51CAA41C EE66B628
DE80774C A4C23D05

u_2Q = 5C9B A4416EAD A45057F6 4ADF29FE B2A6C8D5 7546CEA5
426551DB E4C43157

39B0 51282C27 D6A55E19 CCEDA153 7C02D812 43E65DF8
309E58BC F5030C06

$u_1P + u_2Q = (x_1, y_1)$

x_1 = 6321 0D71EF6C 10157C0D 1053DFF9 3EB8F028 1E3F9DA2
DEB377A8 1BDAE8D5

y_1 = 5EAF D217370E 12036519 CAD381A1 FC38234F 61870DB2
2C1E410A C1F183F0

3. Signature check

a. convert x_1 to an integer $\bar{x}_1$

$\bar{x}_1$ = 6841639825023137355787549028176290563024791328169814824526010005446269 01

b. compute $v = \bar{x}_1 \bmod n$.

v = 2159633321041961198501834003903461262881815148684178964245587692239155 2

c. $v = r'$. OK

H.3. Examples of ECDSA over the Field F_p

H.3.1. An Example With a 192-bit Prime p

a. Elliptic Curve Parameter Setup

1. The field F_p is generated by the prime

p = 6277101735386680763835789423207666416083908700390324961279

2. The curve is $E : y^2 = x^3+ax+b$ over F_p, where

seed = 3045AE6F C8422F64 ED579528 D38120EA E12196D5

r = 3099D2BB BFCB2538 542DCD5F B078B6EF 5F3D6FE2 C745DE65

a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC

b = 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1

3. Generating point P (with point compression)

03 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012

P has prime order

n = 6277101735386680763835789423176059013767194773182842284081

h = 1

b. Key Generation

d = 6510567709060150760568107634563585671901001566956156656 59

$Q = dP = (x_Q, y_Q)$ (with point compression)

02 62B12D60 690CDCF3 30BABAB6 E69763B4 71F994DD 702D16A5

c. Signature Generation

M = "abc"

1. Message digesting

SHA-1 is applied to M to get

e = SHA-1(M)= 968236873715988614170569073515315707566766479517

2. Elliptic curve computation

a. select a k in the interval [2, n-2]

k = 6140507067065001063065065565667405560006161556565665656654

b. compute $R = kP = (x_1, y_1)$

x_1 = 88505238 0FF147B7 34C330C4 3D39B2C4 A89F29B0 F749FEAD

y_1 = 9CF9FA1C BEFEFB91 7747A3BB 29C072B9 289C2547 884FD835

c. convert x_1 to an integer $\bar{x}_1$

$\bar{x}_1$ = 3342403536405981729393488334694600415596881826869351677613

d. set $r = \bar{x}_1 \bmod n$.

r = 3342403536405981729393488334694600415596881826869351677613

e. $r \neq 0$, OK.

3. Modular computation

a. compute $s = k^{-1}(e + dr) \bmod n$.

s = 5735822328888155254683894997897571951568553642892029982342

b. $s \neq 0$, OK.

4. Signature formatting

The signature is the two integers r and s.

r = 3342403536405981729393488334694600415596881826869351677613

s = 5735822328888155254683894997897571951568553642892029982342

d. Signature verification

1. Message digesting

SHA-1 is applied to M' to get

e=SHA-1(M')=968236873715988614170569073515315707566766479517

2. Elliptic curve computation

a. r' is in interval [1, n-1], OK

b. s' is in interval [1, n-1], OK.

c. compute $c = (s')^{-1} \bmod n$

c = 325096440447252682513051649045234621774918970404962904

2861

d. compute $u_1 = ec \bmod n$ and $u_2 = r'c \bmod n$

u_1 = 2563697409189434185194736134579731015366492496392189760599

u_2 = 6266643813348617967186477710235785849136406323338782220568

e. compute $(x_1, y_1) = u_1P + u_2Q$

u_1P = DD9734E5 159253EB 0B09A049 2E12CBA8 7084C11B AC674D82 804F5FDC 638946FA 6660E851 E10542C1 134D4348 2956B50E

u_2Q = 48893A3F 98EBA955 7660BE10 14BBD7D2 42326A1C DA7CF246 114A3118 867D4032 247416C4 A2BA3E83 076B6F8C B666667A

$u_1P + u_2Q = (x_1, y_1)$

x_1 = 88505238 0FF147B7 34C330C4 3D39B2C4 A89F29B0 F749FEAD

y_1 = 9CF9FA1C BEFEFB91 7747A3BB 29C072B9 289C2547 884FD835

3. Signature check

a. convert x_1 to an integer $\bar{x}_1$

$\bar{x}_1$ = 3342403536405981729393488334694600415596881826869351677613

b. compute $v = \bar{x}_1 \bmod n$.

v = 3342403536405981729393488334694600415596881826869351677613

c. v = r'. OK

H.3.2. An Example With a 239-bit Prime *p*

a. Elliptic Curve Parameter Setup

1. The field F_p is generated by the prime

p = 883423532389192164791648750360308885314476597252960362792450860609699839

2. The curve is $E : y^2 = x^3 + ax + b$ over F_p, where

seed = E43BB460 F0B80CC0 C0B07579 8E948060 F8321B7D

r = 28B8 5EC1ECC1 9EFE769E B741A6D1 BA29476A A5A8F261 0957D6EF E78D3783

a = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FFFFFFFF 80000000 00007FFF FFFFFFFC

b = 6B01 6C3BDCF1 8941D0D6 54921475 CA71A9DB 2FB27D1D 37796185 C2942C0A

3. Generating point P (with point compression)

020FFA 963CDCA8 816CCC33 B8642BED F905C3D3 58573D3F
27FBBD3B 3CB9AAAF

P has prime order

n = 883423532389192164791648750360308884807550341691627752
275345424702807307

h = 1

b. Key Generation

d = 876300101507107567501066130761671078357010671067781776
716671676178726717

$Q = dP = (x_Q, y_Q)$ (with point compression)

025B6D C53BC61A 2548FFB0 F671472D E6C9521A 9D2D2534
E65ABFCB D5FE0C70

c. Signature Generation

M = "abc"

1. Message digesting

SHA-1 is applied to M to get

e=SHA-1(M)= 968236873715988614170569073515315707566766479517

2. Elliptic curve computation

a. select a k in the interval [2, n-2]

k = 70000001756905664665550578175715710757050157577570577 9
575555657156756655

b. compute $R = kP = (x_1, y_1)$

x_1 = 2CB7 F36803EB B9C427C5 8D8265F1 1FC50847 47133078
FC279DE8 74FBECB0

y_1 = 20C0 8272B9E6 C92B518A 5AC5EB28 35BE0102 809D77E6
9304A6F7 C522B47B

c. convert x_1 to an integer $\bar{x}_1$

$\bar{x}_1$ = 308636143175167811492622547300668018854959378758531778 1
47462058306432176

d. set $r = \bar{x}_1 \bmod n$.

r = 308636143175167811492622547300668018854959378758531778 1
47462058306432176

e. $r \neq 0$, OK.

3. Modular computation

a. compute $s = k^{-1}(e + dr) \bmod n$.

s = 323813553209797357708078776831250505931891051755007842 7
81978505179448783

b. $s \neq 0$, OK.

4. Signature formatting

The signature is the two integers r and s.

r = 308636143175167811492622547300668018854959378758531778147462058306432176

s = 323813553209797357708078776831250505931891051755007842781978505179448783

d. Signature verification

1. Message digesting

SHA-1 is applied to M' to get

e = SHA-1(M')= 968236873715988614170569073515315707566766479517

2. Elliptic curve computation

a. r' is in interval [1, n-1], OK

b. s' is in interval [1, n-1], OK.

c. compute $c=(s')^{-1} \bmod n$

c = 831843418332978390463010021843350581892480848636408104706147767766249764

d. compute $u_1 = ec \bmod n$ and $u_2 = r'c \bmod n$

u_1 = 124064965052014194622159338097387562954788117638383503089995672152118745

u_2 = 811363736140754465407544341268382421438687214093897850239246340491822539

e. compute $(x_1, y_1) = u_1P+u_2Q$

u_1P =

64C4 29FAF03D	C1707700	D2011D43	9836B4C7	12DCFFD8	E4B7ED99 37F62D1F
6580 DE1A6ECE	DFD78353	8C7C9D83	98BAE8B5	A697EEFD	004AA596 60800F48

u_2Q =

3DCA 0CAFD86C	59DDD9FC	251A2073	9F698451	68F5922E	523B7994 AFC92D9D
5532 B0A717E9	45EED3D8	AD1C26AB	37907E94	2833CD22	AFFE63AC 1F5BC8FE

$u_1P + u_2Q = (x_1, y_1)$

x_1 =

2CB7 F36803EB	B9C427C5	8D8265F1	1FC50847	47133078	FC279DE8 74FBECB0

y_1 =

20C0 8272B9E6	C92B518A	5AC5EB28	35BE0102	809D77E6	9304A6F7 C522B47B

3. Signature check

a. convert x_1 to an integer $\bar{x}_1$

$\bar{x}_1$ = 308636143175167811492622547300668018854959378758531778147462058306432176

b. compute $v = \bar{x}_1 \bmod n$.

v = 308636143175167811492622547300668018854959378758531778147462058306432176

c. $v = r'$. OK

H.4. Sample Elliptic Curves over the Field F_{2^m}

This section presents sample curves over various fields F_{2^m} which may be used to ensure the correct implementation of this Standard.
The curves over the fields $F_{2^{163}}$, $F_{2^{191}}$, $F_{2^{239}}$ and $F_{2^{359}}$ were generated verifiably at random using the method described in Section F.3.3.a.
The curves over the fields $F_{2^{176}}$, $F_{2^{208}}$, $F_{2^{272}}$, $F_{2^{304}}$ and $F_{2^{368}}$ were generated using the Weil Theorem (see Note 6 in Section F.3.2).
The curve over the field $F_{2^{431}}$ was generated at random (but not using the method described in Section F.3.3.a).

H.4.1. 3 Examples With $m = 163$

Elliptic Curve Parameter Setup (pentanomial basis)

1. The field $F_{2^{163}}$ is generated by the irreducible pentanomial

f = 08 00000000 00000000 00000000 00000000 00000107

2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{163}}$.

Example 1.

seed = D2C0FB15 760860DE F1EEF4D6 96E67687 56151754

a = 07 2546B543 5234A422 E0789675 F432C894 35DE5242

b = 00 C9517D06 D5240D3C FF38C74B 20B6CD4D 6F9DD4D9

Generating point P (with point compression)

0307 AF699895 46103D79 329FCC3D 74880F33 BBE803CB

Order of P

n = 04 00000000 00000000 0001E60F C8821CC7 4DAEAFC1

h = 02

Example 2.

seed = 53814C05 0D44D696 E6768756 1517580C A4E29FFD

a = 01 08B39E77 C4B108BE D981ED0E 890E117C 511CF072

b = 06 67ACEB38 AF4E488C 407433FF AE4F1C81 1638DF20

Generating point P (with point compression)

0300 24266E4E B5106D0A 964D92C4 860E2671 DB9B6CC5

Order of P

n = 03 FFFFFFFF FFFFFFFF FFFDF64D E1151ADB B78F10A7

h = 02

Example 3.

seed = 50CBF1D9 5CA94D69 6E676875 615175F1 6A36A3B8

a = 07 A526C63D 3E25A256 A007699F 5447E32A E456B50E

b = 03 F7061798 EB99E238 FD6F1BF9 5B48FEEB 4854252B

Generating point P (with point compression)

0202 F9F87B7C 574D0BDE CF8A22E6 524775F9 8CDEBDCB

Order of P

n = 03 FFFFFFFF FFFFFFFF FFFE1AEE 140F110A FF961309

h = 02

H.4.2. Example With $m = 176$

Elliptic Curve Parameter Setup (pentanomial basis)

1. The field $F_{2^{176}}$ is generated by the irreducible pentanomial

f = 010000 00000000 00000000 00000000 00000800 00000007

2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{176}}$.

Example 1.

seed = NO

a = E4E6 DB299506 5C407D9D 39B8D096 7B96704B A8E9C90B

b = 5DDA 470ABE64 14DE8EC1 33AE28E9 BBD7FCEC 0AE0FFF2

Generating point P (with point compression)

038D16 C2866798 B600F9F0 8BB4A8E8 60F3298C E04A5798

Order of P

n = 01 00925373 97ECA4F6 145799D6 2B0A19CE 06FE26AD

h = FF6E

H.4.3. 5 Examples With $m = 191$

a. Elliptic Curve Parameter Setup (trinomial basis)

1. The field $F_{2^{191}}$ is generated by the irreducible trinomial

f = 80000000 00000000 00000000 00000000 00000000 00000201

2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{191}}$.

Example 1.

seed = 4E13CA54 2744D696 E6768756 1517552F 279A8C84

a = 2866537B 67675263 6A68F565 54E12640 276B649E F7526267

b = 2E45EF57 1F00786F 67B0081B 9495A3D9 5462F5DE 0AA185EC

Generating point *P* (with point compression)

02 36B3DAF8 A23206F9 C4F299D7 B21A9C36 9137F2C8 4AE1AA0D

Order of *P*

n = 40000000 00000000 00000000 04A20E90 C39067C8 93BBB9A5

h = 02

Example 2.

seed = 0871EF2F EF24D696 E6768756 151758BE E0D95C15

a = 40102877 4D7777C7 B7666D13 66EA4320 71274F89 FF01E718

b = 0620048D 28BCBD03 B6249C99 182B7C8C D19700C3 62C46A01

Generating point *P* (with point compression)

02 3809B2B7 CC1B28CC 5A87926A AD83FD28 789E81E2 C9E3BF10

Order of *P*

n = 20000000 00000000 00000000 50508CB8 9F652824 E06B8173

h = 04

Example 3.

seed = E053512D C684D696 E6768756 15175067 AE786D1F

a = 6C010747 56099122 22105691 1C77D77E 77A777E7 E7E77FCB

b = 71FE1AF9 26CF8479 89EFEF8D B459F663 94D90F32 AD3F15E8

Generating point *P* (with point compression)

03 375D4CE2 4FDE4344 89DE8746 E7178601 5009E66E 38A926DD

Order of *P*

n = 15555555 55555555 55555555 610C0B19 6812BFB6 288A3EA3

h = 06

b. Elliptic Curve Parameter Setup (optimal normal basis)

1. The field $F_{2^{191}}$ is generated by the irreducible polynomial

f = D1010001 00000001 00000000 00000001 D1010001 00000001

2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{191}}$.

Example 4.

seed = A399387E AE54D696 E6768756 151750E5 8B416D57

a = 65903E04 E1E49242 53E26A3C 9AC28C75 8BD8184A 3FB680E8

b = 54678621 B190CFCE 282ADE21 9D5B3A06 5E3F4B3F FDEBB29B

Generating point P (with point compression)
02 5A2C69A3 2E8638E5 1CCEFAAD 05350A97 8457CB5F B6DF994A

Order of P
n = 40000000 00000000 00000000 9CF2D6E3 901DAC4C 32EEC65D

h = 02

Example 5.

seed = 2D88F7BC 545794D6 96E67687 56151759 73391555

a = 25F8D06C 97C82253 6D469CD5 170CDD7B B9F500BD 6DB110FB

b = 75FF570E 35CA94FB 3780C261 9D081C17 AA59FBD5 E591C1C4

Generating point P (with point compression)
03 2A16910E 8F6C4B19 9BE24213 857ABC9C 992EDFB2 471F3C68

Order of P
n = 0FFFFFFF FFFFFFFF FFFFFFFF EEB354B7 270B2992 B7818627

h = 08

H.4.4. Example With $m = 208$

Elliptic Curve Parameter Setup (pentanomial basis)

1. The field $F_{2^{208}}$ is generated by the irreducible pentanomial

f = 010000 00000000 00000000 00000000 00080000 00000000 00000007

2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{208}}$.

Example 1.

seed = NO

a = 0000 00000000 00000000 00000000 00000000 00000000 00000000

b = C861 9ED45A62 E6212E11 60349E2B FA844439 FAFC2A3F D1638F9E

Generating point P (with point compression)

0289FD FBE4ABE1 93DF9559 ECF07AC0 CE78554E 2784EB8C
1ED1A57A

Order of P

n = 01 01BAF95C 9723C57B 6C21DA2E FF2D5ED5 88BDD571
7E212F9D

h = FE48

H.4.5. 5 Examples With $m = 239$

a. Elliptic Curve Parameter Setup (trinomial basis)

1. The field $F_{2^{239}}$ is generated by the irreducible trinomial

f = 8000 00000000 00000000 00000000 00000000 00000000
00000010 00000001

2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{239}}$.

Example 1.

seed = D34B9A4D 696E6768 75615175 CA71B920 BFEFB05D

a = 3201 0857077C 5431123A 46B80890 6756F543 423E8D27
87757812 5778AC76

b = 7904 08F2EEDA F392B012 EDEFB339 2F30F432 7C0CA3F3
1FC383C4 22AA8C16

Generating point P (with point compression)
025792 7098FA93 2E7C0A96 D3FD5B70 6EF7E5F5 C156E16B
7E7C8603 8552E91D

Order of P

n = 2000 00000000 00000000 00000000 000F4D42 FFE1492A
4993F1CA D666E447

h = 04

Example 2.

seed = 2AA6982F DFA4D696 E6768756 15175D26 6727277D

a = 4230 017757A7 67FAE423 98569B74 6325D453 13AF0766
266479B7 5654E65F

b = 5037 EA654196 CFF0CD82 B2C14A2F CF2E3FF8 775285B5
45722F03 EACDB74B

Generating point P (with point compression)
0228F9 D04E9000 69C8DC47 A08534FE 76D2B900 B7D7EF31
F5709F20 0C4CA205

Order of P

n = 1555 55555555 55555555 55555555 553C6F28 85259C31
E3FCDF15 4624522D

h = 06

Example 3.

seed = 9E076F4D 696E6768 75615175 E11E9FDD 77F92041

a = 0123 8774666A 67766D66 76F778E6 76B66999 176666E6
87666D87 66C66A9F

b = 6A94 1977BA9F 6A435199 ACFC5106 7ED587F5 19C5ECB5
41B8E441 11DE1D40

Generating point P (with point compression)
0370F6 E9D04D28 9C4E8991 3CE3530B FDE90397 7D42B146
D539BF1B DE4E9C92

Order of P
n = 0CCC CCCCCCCC CCCCCCCC CCCCCCCC CCAC4912 D2D9DF90
3EF9888B 8A0E4CFF

h = 0A

b. Elliptic Curve Parameter Setup (optimal normal basis)

1. The field $F_{2^{239}}$ is generated by the irreducible polynomial

f = D1010001 D1010000 00000001 D1010000 00000000 00000000
00000001 D1010000

2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{239}}$.

Example 4.

seed = F851638C FA4D696E 67687561 51755651 3841BFAC

a = 182D D45F5D47 0239B898 3FEA47B8 B292641C 57F9BF84
BAECDE8B B3ADCE30

b = 147A 9C1D4C2C E9BE5D34 EC02797F 76667EBA D5A3F93F
A2A524BF DE91EF28

Generating point P (with point compression)
034912 AD657F1D 1C6B32ED B9942C95 E226B06F B012CD40
FDEA0D72 197C8104

Order of P
n = 2000 00000000 00000000 00000000 00474F7E 69F42FE4
30931D0B 455AAE8B

h = 04

Example 5.

seed = 2C04F44D 696E6768 75615175 C586B41F 6CA150C9

a = 1ECF 1B9D28D8 017505E1 7475D3DF 2982E243 CA5CB5E9
F94A3F36 124A486E

b = 3EE2 57250D1A 2E66CEF2 3AA0F25B 12388DE8 A10FF955
4F90AFBA A9A08B6D

Generating point *P* (with point compression)
021932 79FC543E 9F5F7119 189785B9 C60A249B E4820BAF
6C24BDFA 2813F8B8

Order of *P*
n = 1555 55555555 55555555 55555555 558CF77A 5D0589D2
A9340D96 3B7AD703

h = 06

H.4.6. Example With *m* = 272

Elliptic Curve Parameter Setup (pentanomial basis)

1. The field $F_{2^{272}}$ is generated by the irreducible pentanomial

f = 010000 00000000 00000000 00000000 00000000 00000000
00000000 01000000 0000000B

2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{272}}$.

Example 1.

seed = NO

a = 91A0 91F03B5F BA4AB2CC F49C4EDD 220FB028 712D42BE
752B2C40 094DBACD B586FB20

b = 7167 EFC92BB2 E3CE7C8A AAFF34E1 2A9C5570 03D7C73A
6FAF003F 99F6CC84 82E540F7

Generating point *P* (with point compression)
026108 BABB2CEE BCF78705 8A056CBE 0CFE622D 7723A289
E08A07AE 13EF0D10 D171DD8D

Order of *P*
n = 01 00FAF513 54E0E39E 4892DF6E 319C72C8 161603FA
45AA7B99 8A167B8F 1E629521

h = FF06

H.4.7. Example With *m* = 304

Elliptic Curve Parameter Setup (pentanomial basis)

1. The field $F_{2^{304}}$ is generated by the irreducible pentanomial

f = 010000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000807

2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{304}}$.

Example 1.

seed = NO

a = FD0D 693149A1 18F651E6 DCE68020 85377E5F 882D1B51
0B441600 74C12880 78365A03 96C8E681

b = BDDB 97E555A5 0A908E43 B01C798E A5DAA678 8F1EA279
4EFCF571 66B8C140 39601E55 827340BE

Generating point P (with point compression)
02197B 07845E9B E2D96ADB 0F5F3C7F 2CFFBD7A 3EB8B6FE
C35C7FD6 7F26DDF6 285A644F 740A2614

Order of P
n = 01 01D55657 2AABAC80 0101D556 572AABAC 8001022D
5C91DD17 3F8FB561 DA689916 4443051D

h = FE2E

H.4.8. Example With $m = 359$

Elliptic Curve Parameter Setup (trinomial basis)

1. The field $F_{2^{359}}$ is generated by the irreducible trinomial

f = 80 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000010 00000000 00000001

2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{359}}$.

Example 1.

seed = 2B354920 B724D696 E6768756 1517585B A1332DC6

a = 56 67676A65 4B20754F 356EA920 17D94656 7C466755
56F19556 A04616B5 67D223A5 E05656FB 549016A9 6656A557

b = 24 72E2D019 7C49363F 1FE7F5B6 DB075D52 B6947D13
5D8CA445 805D39BC 34562608 9687742B 6329E706 80231988

Generating point P (with point compression)
033C 258EF304 7767E7ED E0F1FDAA 79DAEE38 41366A13
2E163ACE D4ED2401 DF9C6BDC DE98E8E7 07C07A22 39B1B097

Order of P
n = 01 AF286BCA 1AF286BC A1AF286B CA1AF286 BCA1AF28
6BC9FB8F 6B85C556 892C20A7 EB964FE7 719E74F4 90758D3B

h = 4c

H.4.9. Example With $m = 368$

Elliptic Curve Parameter Setup (pentanomial basis)

1. The field $F_{2^{368}}$ is generated by the irreducible pentanomial

f = 010000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00200000 00000000 00000007

2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{368}}$.

Example 1.

seed = NO

a = E0D2 EE250952 06F5E2A4 F9ED229F 1F256E79 A0E2B455
970D8D0D 865BD947 78C576D6 2F0AB751 9CCD2A1A 906AE30D

b = FC12 17D4320A 90452C76 0A58EDCD 30C8DD06 9B3C3445
3837A34E D50CB549 17E1C211 2D84D164 F444F8F7 4786046A

Generating point *P* (with point compression)

021085 E2755381 DCCCE3C1 557AFA10 C2F0C0C2 825646C5
B34A394C BCFA8BC1 6B22E7E7 89E927BE 216F02E1 FB136A5F

Order of *P*

n = 01 0090512D A9AF72B0 8349D98A 5DD4C7B0 532ECA51
CE03E2D1 0F3B7AC5 79BD87E9 09AE40A6 F131E9CF CE5BD967

h = FF70

H.4.10. Example With *m* = 431

Elliptic Curve Parameter Setup (trinomial basis)

1. The field $F_{2^{431}}$ is generated by the irreducible trinomial

f = 8000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 01000000 00000000
00000000 00000001

2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{431}}$.

Example 1.

seed = NO

a = 1A82 7EF00DD6 FC0E234C AF046C6A 5D8A8539 5B236CC4
AD2CF32A 0CADBDC9 DDF620B0 EB9906D0 957F6C6F EACD6154
68DF104D E296CD8F

b = 10D9 B4A3D904 7D8B1543 59ABFB1B 7F5485B0 4CEB8682
37DDC9DE DA982A67 9A5A919B 626D4E50 A8DD731B 107A9962
381FB5D8 07BF2618

Generating point *P* (with point compression)

02120F C05D3C67 A99DE161 D2F40926 22FECA70 1BE4F50F
4758714E 8A87BBF2 A658EF8C 21E7C5EF E965361F 6C2999C0
C247B0DB D70CE6B7

Order of *P*

n = 03 40340340 34034034 03403403 40340340 34034034
03403403 40340323 C313FAB5 0589703B 5EC68D35 87FEC60D
161CC149 C1AD4A91

h = 2760

H.5. Sample Elliptic Curves over the Field F_p

This section presents sample curves over 192-bit, 239-bit and 256-bit prime fields F_p which may be used to ensure the correct implementation of this Standard.

The curves were generated verifiably at random using the method described in Section F.3.3.b.

H.5.1. 3 Examples With a 192-bit Prime

Elliptic Curve Parameter Setup

1. The field F_p is generated by the prime

p = 6277101735386680763835789423207666416083908700390324961279

2. The curve is $E: y^2 = x^3+ax+b$ over F_p.

Example 1.

seed = 3045AE6F C8422F64 ED579528 D38120EA E12196D5

r = 3099D2BB BFCB2538 542DCD5F B078B6EF 5F3D6FE2 C745DE65

a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC

b = 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1

Generating point P (with point compression)

03 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012

Order of P

n = FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831

h = 01

Example 2.

seed = 31A92EE2 029FD10D 901B113E 990710F0 D21AC6B6

r = 15038D1D 2E1CAFEE 0299F301 1C1DC75B 3C2A86E1 35DB1E6B

a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC

b = CC22D6DF B95C6B25 E49C0D63 64A4E598 0C393AA2 1668D953

Generating point P (with point compression)

03 EEA2BAE7 E1497842 F2DE7769 CFE9C989 C072AD69 6F48034A

Order of P

n = FFFFFFFF FFFFFFFF FFFFFFFE 5FB1A724 DC804186 48D8DD31

h = 01

Example 3.

seed = C4696844 35DEB378 C4B65CA9 591E2A57 63059A2E

r = 25191F95 024D8395 46D9A337 5639A996 7D52F137 3BC4EE0B

a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC

b = 22123DC2 395A05CA A7423DAE CCC94760 A7D46225 6BD56916

Generating point P (with point compression)

02 7D297781 00C65A1D A1783716 588DCE2B 8B4AEE8E
228F1896

Order of P

n = FFFFFFFF FFFFFFFF FFFFFFFF 7A62D031 C83F4294 F640EC13

h = 01

H.5.2. 3 Examples With a 239-bit Prime Elliptic Curve Parameter Setup

1. The field F_p is generated by the prime

p = 883423532389192164791648750360308885314476597252960362
792450860609699839

2. The curve is $E: y^2 = x^3+ax+b$ over F_p.

Example 1.

seed = E43BB460 F0B80CC0 C0B07579 8E948060 F8321B7D

r = 28B8 5EC1ECC1 9EFE769E B741A6D1 BA29476A A5A8F261
0957D6EF E78D3783

a = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FFFFFFFF 80000000
00007FFF FFFFFFFC

b = 6B01 6C3BDCF1 8941D0D6 54921475 CA71A9DB 2FB27D1D
37796185 C2942C0A

Generating point P (with point compression)

020FFA 963CDCA8 816CCC33 B8642BED F905C3D3 58573D3F
27FBBD3B 3CB9AAAF

Order of P

n = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FF9E5E9A 9F5D9071
FBD15226 88909D0B

h = 01

Example 2.

seed = E8B40116 04095303 CA3B8099 982BE09F CB9AE616

r = 1DF4 91E44E7C CAF4D1EA D8A6B90D AE09E0D3 3F2C6CFE
7A6BA76E 86713D52

a = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FFFFFFFF 80000000
00007FFF FFFFFFFC

b = 617F AB683257 6CBBFED5 0D99F024 9C3FEE58 B94BA003
8C7AE84C 8C832F2C

Generating point P (with point compression)

0238AF 09D98727 705120C9 21BB5E9E 26296A3C DCF2F357
57A0EAFD 87B830E7

Order of P

n = 7FFF FFFFFFFF FFFFFFFF FFFF8000 00CFA7E8 594377D4
14C03821 BC582063

h = 01

Example 3.

seed = 7D737416 8FFE3471 B60A8576 86A19475 D3BFA2FF

r = 3A4F 9DC9A6CE FD5F9D11 93B9C996 8C202430 003C2819
C2E49861 8DC58330

a = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FFFFFFFF 80000000
00007FFF FFFFFFFC

b = 2557 05FA2A30 6654B1F4 CB03D6A7 50A30C25 0102D498
8717D9BA 15AB6D3E

Generating point P (with point compression)

036768 AE8E18BB 92CFCF00 5C949AA2 C6D94853 D0E660BB
F854B1C9 505FE95A

Order of P

n = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FF975DEB 41B3A605
7C3C4321 46526551

h = 01

H.5.3. 1 Example With a 256-bit Prime Elliptic Curve Parameter Setup

1. The field F_p is generated by the prime

p = 1157920892103562487626974469494075735300861434152903141955336313088670978539 51

2. The curve is $E: y^2 = x^3+ax+b$ over F_p.

Example 1.

seed = C49D3608 86E70493 6A6678E1 139D26B7 819F7E90

r = 7EFBA166 2985BE94 03CB055C 75D4F7E0 CE8D84A9 C5114ABC
AF317768 0104FA0D

a = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF
FFFFFFFF FFFFFFFC

b = 5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6
3BCE3C3E 27D2604B

Generating point $P = (x, y)$ (without point compression)

x = 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0
F4A13945 D898C296

y = 4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5

Order of P

n = FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551

h = 01

Appendix I Small Example of the ECDSA
[Informative]

I.1. System Setup

The underlying finite field is F_{23}, and the elliptic curve is $y^2 = x^3 + x + 1$, as described in Example 5 in Section C.3. The point $P = (x_P, y_P) = (13, 7)$ is selected. Since $7P = \mathcal{O}$, the point P has order $n = 7$.

The system parameters (the public information) are:

- the field F_{23},
- the curve E,
- the point P, and
- the order $n = 7$
- the cofactor $h = 4$.

I.2. Key Generation

Entity A performs the following operations.

1. A selects a random integer $d = 3$ in the interval $[2, n - 2] = [2, 5]$.
2. A computes the point $Q = dP = 3(13, 7) = (17, 3)$.
3. A makes public the point Q.
4. A′s private key is the integer $d = 3$.

I.3. Signature Generation for ECDSA

Entity A signs message M = 11100011010111100.

Suppose that the decimal representation of the hash value $H(M)$ is $e = 6$.

Entity A:

1. selects a random integer $k = 4$ in the interval $[2, n - 2] = [2, 5]$.
2. computes:
   $(x_1, y_1) = kP$
   $= 4(13, 7)$
   $= (17, 20)$.
3. represents x_1 as the integer $\bar{x}_1 = 17$.
4. sets $r = \bar{x}_1 \bmod n = 17 \bmod 7 = 3$.
5. computes:
   $s = k^{-1}(e + dr) \bmod n$
   $= 4^{-1}(6 + 3 \cdot 3) \bmod 7$
   $= 2(15) \bmod 7$
   $= 2$.

The signature on message M is $(r, s) = (3, 2)$.

I.4. Signature Verification for ECDSA

Entity B verifies signature $(r', s') = (3, 2)$ on M as follows.

Entity B:

1. looks up A′s public key $Q = (17, 3)$.
2. computes $\bar{e} = 7$, the decimal representation of $H(M)$.

3. computes:

   $c = (s')^{-1} \bmod n$
   $= 2^{-1} \bmod 7$
   $= 4.$

4. computes

   $u_1 = ec \bmod n$
   $= 6 \cdot 4 \bmod 7$
   $= 3$

   and

   $u_2 = r'c \bmod n$
   $= 3 \cdot 4 \bmod 7$
   $= 5.$

1. computes the point:

   $(x_1, y_1) = u_1P + u_2Q = 3P + 5Q = 3(13, 7) + 5(17, 3) = (17, 20).$

6. represents x_1 as the integer $\bar{x}_1 = 17$.

7. computes $v = \bar{x}_1 \bmod n = 17 \bmod 7 = 3$.

8. accepts the signature since $v = r' = 3$.